Process execution in Cyber-Physical Systems using Cloud and Cyber-Physical Internet services

Borja Bordel [1,*], Ramón Alcarria [2], Diego Sánchez de Rivera [1], Tomás Robles [1]

[1] Department of Telematics Systems Engineering. Universidad Politécnica de Madrid. Avenida Complutense nº 30. 28040 - Madrid (España); E-Mails: bbordel@dit.upm.es; diego.sanchezderiveracordoba@gmail.com; tomas.robles@upm.es;

[2] Department of Topographic Engineering and Cartography. Universidad Politécnica de Madrid. Campus Sur, 28031 Madrid, Spain; E-Mail: ramon.alcarria@upm.es

***** Author to whom correspondence should be addressed; E-Mail: bbordel@dit.upm.es; Tel. 91 549 57 00 ext. 3035

ABSTRACT

"Cyber-Physical Systems" (CPS) have emerged as the next technological revolution. These new systems are commonly supported by a collection of ad hoc connected devices which typically collaborate in order to control some physical processes. However, in recent years, many other applications based on the CPS paradigm have been reported. In particular, executing user-defined processes over a cyber-physical infrastructure is a very promising technology for the future. Therefore, in this paper, we propose a scheme which allows the creation of user-defined processes, their decomposition and translation into executable orders or code, and their execution using the locally available cyber-physical infrastructure, cloud services and/or other services offered by remote CPS through the Cyber-Physical Internet. The proposed solution also enables the execution of processes with a guaranteed QoS. Moreover, an experimental validation is provided in order to evaluate the proposed technology performance. In particular, it is proved that more than 95% of processes are correctly executed, and only in a 2% of cases the minimum cost execution is not selected.

KEYWORDS

Cyber-Physical Systems; Workflow execution; Cyber-physical process modeling; Cyber-Physical internet; Cloud services; Process execution; YAWL;

FUNDING

The research leading to these results has received funding from the Ministry of Economy and Competitiveness through SEMOLA project (TEC2015-68284-R) and from the Autonomous Region of Madrid through MOSI-AGIL-CM project (grant P2013/ICE-3019, co-funded by EU Structural Funds FSE and FEDER). Borja Bordel has received funding from the Ministry of Education through the FPU program (grant number FPU15/03977)

1. INTRODUCTION

The emerging term “Cyber-Physical Systems” refers to the integration of computational and physical capabilities using ad hoc sensor and actuator networks which, typically, are seamlessly integrated into daily living objects [25]. However, in recent years (since 2014, approximately), works about CPS also consider other devices such as microcontrollers [18], legacy systems [14] or even humans [44], which greatly extend the system’s capabilities. Based on these improved CPS, new applications, architectures and functionalities have been reported [51][48]. Moreover, these new paradigm has been also employed to develop solutions in traditional fields such as the ad hoc and wireless sensor networks, Internet of Things (IoT) or pervasive computing [12].

In this context, one of the most promising proposals is the so-called service-oriented CPS [24]. In this kind of CPS, the physical infrastructure offers a collection of services by means of some software elements (such as service composition engines [2]). These services may be used to perform a certain fixed application (deployed by technological experts), or might be included in a prosumer [37] environment where non-expert users can design, build, deploy and remove their own applications in a dynamic way.

In a particular realization of the second case, services are employed to execute user-defined processes. In these processes, services are invoked and linked with each other in a certain way, in order to perform a collection of tasks or activities. Processes may describe business tasks, control activities or assisted living policies among other possibilities. Besides, each process can be executed once, various times, when a certain event happens, periodically, etc. Furthermore, most times, processes have to include services which are not supported by the underlying physical infrastructure. Sometimes these services are provided by external companies (such as the bank services), other times services are provided from a remote location (for example, if the company manages various production plants) and other services are supported by a virtual infrastructure. In the first and third case cloud services are employed [53]; in the second case, services provided by remote CPS might be used if the systems are connected to the so-called Cyber-Physical Internet [32][21] -CPI-. CPI is a network of CPS which publicly offers services that can be invoked remotely.

In order to enable non-expert users to define their processes, services must be described in a high level of abstraction. Then, instruments to decompose and translate users’ designs into executable code and actions are required, as well as tools for controlling the execution and the temporization. However, current proposals only cover this objective partially. Thus, the main contribution of this work is to define a scheme being able to perform these tasks using new and emerging technologies such as the Cyber-Physical Systems paradigm.

Therefore, the objective of this paper is to describe a technical solution, including a functional architecture and the required algorithms, which enables users to define their own processes, which locates the selected services locally, in the cloud or in the CPI (selecting the minimum cost location if various are available), and being able to decompose users’ designs into executable code and control the execution and the temporization. Additionally, the proposed solution allows users to perform executions with a guaranteed Quality-of-Service (QoS).

The rest of the paper is organized as follows. Section 2 analyzes the state of the art on task and process execution, delegation and transformation in CPS. Section 3 presents the proposal, including the reference architecture, the functional architecture and the required algorithms and

mathematical formalization. Section 4 explains the experimental validation and Section 5 presents the results. Finally, Section 6 concludes the paper.

2. STATE OF THE ART ON PROCESS AND TASK EXECUTION, DELEGATION AND TRANSFORMATION IN CPS

Traditional implementations of CPS try to relate physical and computational processes [23][9]. Thus, in these systems, computational processes are defined as state machines where transitions between states depend on the physical processes [15]. The system behavior in each state, as well as the transitions, is defined at very low level (fixing the actions performed for each device) so, typically, these systems act as closed systems only manipulated by technological experts. A possible generalization of this solution [7] not only considers physical processes, but also any other event which can trigger a transition (for example, an alarm). Even more general proposals, such as the named *Elastic Systems* [28], report the integration of people in those processes. In this solution, however, domain experts continue having problems to define their own processes as it is mandatory to know how the system works at very low level.

In order to allow domain experts to define their own processes for CPS, more modern proposals have included the concept of "task" [34] instead of the state machines. A task represents a collection of actions to be performed in order to obtain a certain output. Relating the outputs and the inputs of the different tasks, a global process may be defined. A task can be described in many different abstraction levels, from human language to binary code. Thus, some proposals [42] describe a scheme where various translation (or transformation) engines (typically three cascaded elements) turn a high-level description of a process into a machine-level description to be executed by the underlying devices.

Most usually employed transformation techniques nowadays are based on a set of transformation rules which are applied by a software agent [13]. In many cases, this agent takes the form of a compiler [3], sometimes implementing advanced solutions such as predictive compilation [10]. In the last years, besides, semantic solutions have been proposed and successfully deployed [43]. Finally, specific solutions are also available. For example, it may be found transformation engines for embedded microcontrollers [6].

Previously described proposals, however, present a limited usefulness, mainly because of two facts: first, metadata cannot be easily specified and, second, standard description languages are not developed enough. In order to address the lack of adequate process description languages (especially medium-level languages) several authors have tried to extend existing standard solutions such as BPMN [13][41] or BPEL [3][45]. Nevertheless, these proposals do not solve the problem, as no execution, transformation engine, or any other tool, is adapted or extended to these proposals. In respect to the metadata specification, temporization is probably the most important issue. In all previously mentioned works, temporization must be included (when possible) using complicated notations as it is not natively supported. Then, papers proposing various schedulers for CPS temporization control have been proposed [56][19]. However, the integration of these schedulers with the other elements is not clear at all.

In order to address this problem, researchers on ad hoc wireless sensor networks (and on CPS) have proposed to describe tasks not as a collection of actions, but as a sequence of services which must be invoked [24][22]. Then, an orchestrator (or central execution engine) manages

the synchronization among the different services in order to obtain a coherent result [38]. Some proposals even describe management algorithms which adapt the evolution of the orchestration policies depending on the variable external physical processes [27]. These solutions allow specifying temporization conditions in a very easy way in the orchestrator, although other problems (such as the need of knowing the system behavior at very low level) remain.

On the other hand, various works on CPS consider the idea of including not only services offered from the local physical platform, but also from the cloud [1] or from remote installation through the Internet [49] (named as Cyber-Physical Internet or Internet of CPS when connects various CPS). In all these works, however, the location of every service is stablished by default, and for every service only one location is available.

In any case, if various locations where tasks might be executed are available, a task delegation algorithm must be implemented (in order to transfer tasks to the system in charge of executing them). In the simplest task delegation algorithm, the root agent turns on a certain secondary node when a task (or various tasks) is being delegated [11]. In this solution, each node works independently. The root node suppresses the task from its execution schedule when the task is delegated to his secondary node, which is programmed to perform this task. This solution is the simplest and easiest to implement and deploy; however, is a very rigid scheme which hardly adapts to changes in the platform. In a more advanced technique, nodes are software agents. These agents offer public methods to be invoked by other agents paying back the agreed "price" [31]. Using this platform, agents may delegate parts of the execution in a very efficient way. Nevertheless, problems such as object serialization or method publication have not been totally resolved. Finally, service-oriented solutions have been proposed. In these works, various platforms share information to discover the list of available services, and to order the platform to execute a certain task or collection of them [35]. In this case all the available technologies for distributed systems and cloud-computing may be used (serialization, coherence, etc.). To improve the dynamism of the service-oriented solutions they might be complemented with an algorithm to determine the most adequate location to execute the tasks (if various available) [16]. These algorithms are based on cost functions, whose parameters may vary dynamically.

The objective of this paper is to go beyond the current technological state, proposing the first unified service-oriented process execution scheme, based on the emerging CPS. Our proposal considers a service-oriented solution, where various transformation engines allow users to describe processes in a very high abstraction level and, later, translate these designs into executable code. In order to employ standard modules and techniques, a well-known standard process description language such as YAWL [47] is used in our proposal. As a novelty, two different execution engines are considered, in order to interact with the cloud and the emerging Cyber-Physical Internet and, at the same time, wean the particularities of the physical devices and the high-level services. In our solution, a service may be offered in different locations and the most convenient location to execute it is selected dynamically. The task, then, could be delegated. Moreover, also as a novelty, together with every process description file, a metadata description file can be uploaded to the execution engines, contributing to control the temporization, the QoS and other parameters.

3. PROPOSED SOLUTION

In this Section the technical solution is detailed. In the first subsection the reference and a general description of the solution are presented. And, in the second subsection, the technical proposal is described in detail.

3.1 ARCHITECTURES, PREVIOUS CONFIGURATIONS, OVERVIEW OF THE PROPOSAL

The main problem to address when designing a solution for process execution in CPS is the need of allowing users to describe processes in a high level of abstraction and, later, translate them into instructions compatible with devices.

In order to address this problem, we are basing our solution in a reference architecture (see Figure 1) proposed by the National Institute of Standards and Technology (NIST) which represents a CPS as a stack of six abstraction levels [7].

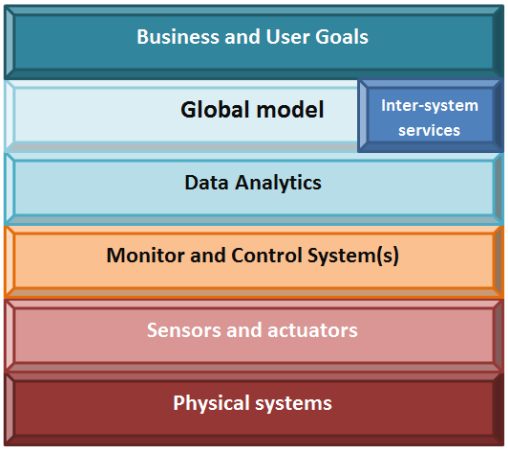


Figure 1: Reference architecture for CPS

In Figure 1, the first layer is dedicated to *Business and user goals*. That includes the process definition in a human understandable language and the definition of the objectives of the defined processes. In *Global model*, a global first executable representation of the processes must be obtained. In this layer each service must be located locally, in the cloud or in other Internet locations (the last two elements are known as *Inter-system services*), selecting the most convenient if available in various locations. *Data analytics* layer weans the higher layers from the physical devices' low-level details. For that it implements technologies such as pattern recognition or knowledge extraction. Finally, *Monitor and control systems* layer is dedicated to manage the hardware devices represented by the *Sensors and actuators* layer. *Physical systems* layer refers to the environment which surrounds the CPS.

Considering the division of the functionalities in a CPS proposed by the NIST (see Figure 1), a unified process execution scheme for CPS may be described as a sequence of five steps: process design, business execution, probabilistic execution, hardware access and hardware execution. Figure 2 presents a graphic representation for this "lifecycle", identifying each step with the reference layer (Figure 1) which contains the functionalities employed to perform the actions included in the step.

The first step, *process design*, includes all actions to obtain executable process description. In this step prosumer users define their own processes in any domain language, including natural English and graphic methods. Besides, specialized instruments translate the description provided by prosumer users into a standard workflow description language, such as YAWL [56]. Finally, files containing metadata and an executable description of the process at business level (disaggregating the prosumer services into collections of business services) are generated.

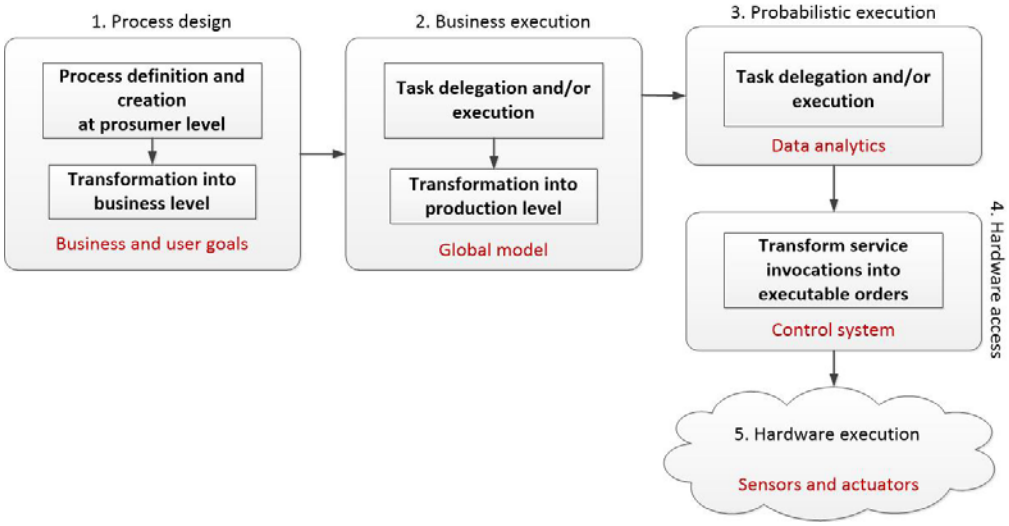


Figure 2: Lifecycle of a process in the proposed solution

In the second step, *business execution*, most important actions in the system are performed. At this step the process execution is initiated and orchestrated. For each service it is calculated where to execute each service task, considering the available locations for each service, the cost, quality, and the user's restrictions for each location (described in the metadata file). Tasks to be delegated are transmitted to the appropriate remote system using the delegation services. The description files of the tasks to be executed locally are translated into a production level description, considering the underlying hardware platform. Metadata are also transformed in the appropriate way.

The third step, *probabilistic execution*, encapsulates all actions intended (1) to execute and orchestrate the processes at production level; and (2) to wean the hard definition of computational processes from the aleatory behavior of hardware. For example, sensors present an intrinsic error which must be taken into account; various measurements made in a row offer different values, etc.

In the fourth step, *hardware access*, generic service invocations done by the execution engines are translated into the execution orders required by the underlying ad hoc network. In order to allow the inclusion of virtual devices [26] in the ad hoc network, at this step services may also be redirected to the cloud, if the service is not supported by real devices but virtual.

Finally, in the fifth step, *hardware execution*, the execution actions are processed by underlying network of physical and virtual devices which are ad hoc connected [50]. Reports about the results are also generated.

3.2 DETAILED EXPLANATION OF THE PROPOSAL

In this Section we are describing in detail the proposed solution, algorithms and data format employed in the first four steps showed in Figure 2: process design, business execution, probabilistic execution and hardware access. The last step (hardware execution) depends on the underlying ad hoc network, and contributing to this process is not the objective of this paper (Section 3.2.4 describes some details).

3.2.1 Process design

Figure 3 presents a detailed description of the functional components involved in the process design. At this step, processes and tasks are based on prosumer services. These services are obtained as a composition of business services (see Section 3.2.2), cloud services and Internet services (which are business services provided by remote CPS). Prosumer services are described in a high level and are human-understandable, in order to allow prosumer users to define their own processes. The catalogue of available prosumer services is maintained in the *prosumer services repository*. Apart from the list of services no additional information is required to be included in this repository. Methods to calculate and maintain this catalogue of available services are very varied [30]: manual, automatic [8], based on sematic technologies, on messaging, etc. Any of them is valid and applicable to our proposal, which is independent from the system configuration.

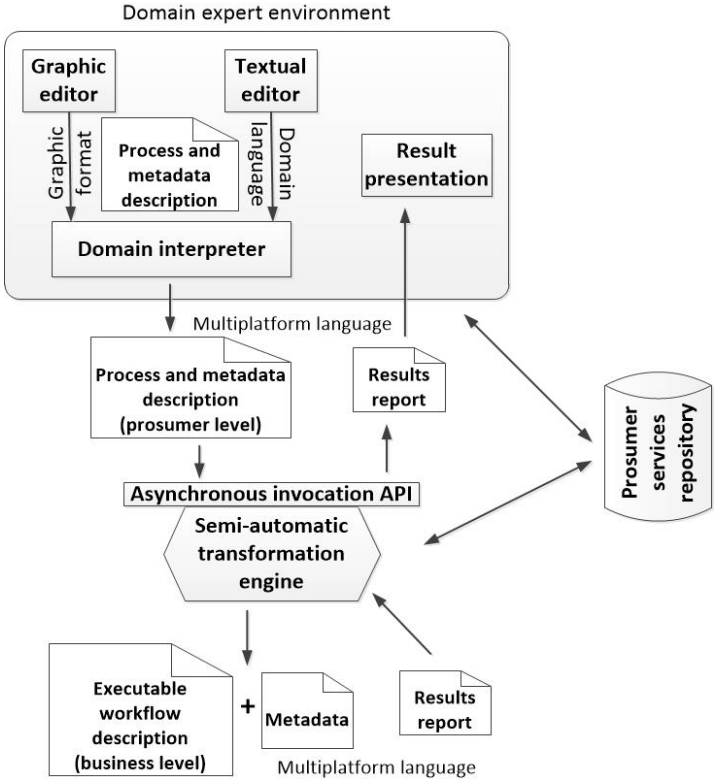


Figure 3: Detailed functional architecture for process design

Once, information about the prosumer services is available in the corresponding repository, and using the tools provided in the *domain expert environment*, prosumer users may design their own processes to be executed in the CPS as a logical sequence (or workflow) of prosumer services.

Table 1: Default information for mandatory metadata

Metadata	Default information
Temporization	The process execution starts immediately (when the user applies for it), it is executed only once and each task is executed only once too.
Execution restrictions	Every task can be executed in any location (cloud, Internet or local)
Required QoS	No guaranteed Quality-of-Service is required

```
(define-parameter
  :variable ?start_hour
  :value (?Calendar.HOUR_OF_DAY)
)
(define-parameter
  :variable ?process-repetitions
  :value 3
)
(define-parameter
  :variable ?start_hour_prosumerTask#2
  :value (?Calendar.HOUR_OF_DAY)
)
/* Other temporization metadata */
(define-parameter
  :variable ?execution-restriction-prosumerTask#4
  :value (?URI.local)
)
/* Other quality or execution restrictions metadata */
```

```
/* Auxiliary variables */
(define-activity-role
  :id ProsumerTask#1
  :name prosumerService.one
  :successors 2
  :preconditions (AND InputCondition)
  :postconditions
    (ProsumerCondition#1 OR  ProsumerCondition#2)
)
/* Other task definitions */
(define-activity-role
  :id ProsumerTask#5
  :name prosumerService.five
  :successors 0
  :preconditions (AND ProsumerTask#5 (endof))
  :postconditions (XOR OutputCondition)
)
```

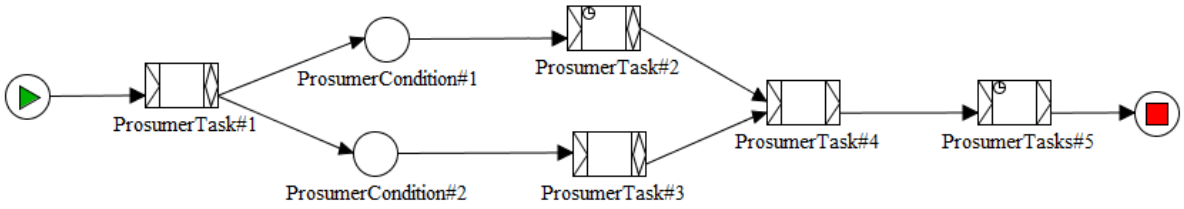


Figure 4: Domain-dependent process description (a) Textual using PSL (b) Graphic using YAWL

In order to create a process, prosumer users are provided with two different tools: a textual editor and a graphic editor. Using the textual editor, users can describe processes employing the most adequate domain language; such as the Process Specification Language (PSL) [33] in manufacturing scenarios or, even, plain English if preferred. Figure 4(a) presents an example of the output of the textual editor. As can be seen, metadata must be also included in that description. Many parameters can be included as metadata, however, information about three topics is considered mandatory: temporization, execution restrictions and required QoS. Prosumer users may indicate information about these topics in a high-level of abstraction, being later translated into different parameters at different levels (business, production, etc.). If no

information for these topics is included, some information is aggregated by default (see Table 1).

Although textual editor enables users to employ their domain language, prosumer environments are commonly based on graphic design. Using the graphic editor, users may create processes based on the available prosumer services without knowing any domain description language. Technologies such as YAWL (which we are employing) or BPMN can be employed at this point. Figure 4(b) presents the output of the graphic editor for the same process described in Figure 4(a). In this case, metadata are introduced in the system using the same graphic interface, although this information is not presented in the process representation.

In the process descriptions provided by users, tasks are completely executed by invoking only one prosumer service for each task. Thus, using the YAWL terminology, in those description files tasks are considered as atomic, as no secondary decomposition (labeled with the tag *NetFactsType* in YAWL) for the tasks is explicitly included in the file, see Figure 4, although prosumer services could be decomposed into various business services. Graphically, that is indicated in Figure 4(b) by employing a simple square for representing tasks.

```
<!- - XML  YAWL header - ->
<decomposition id="ProsumerProcess" isRootNet="true" xsi:type="NetFactsType">
    <!- - Other local variables declaration (metadata) - ->
   <localVariable>
    <index>1</index>
    <name> process-repetitions </name>
    <type>Integer</type>
    <namespace>http://www.w3.org/2001/XMLSchema</namespace>
     <initialValue>3</initialValue>
   </localVariable>
   <!- - Other local variables declaration (metadata) and auxiliary variables- ->
   <processControlElements>
      <!- - Other task or condition declarations - ->
      <task id="ProsumerTask1">
          <name>ProsumerTask#1</name>
          <flowsInto>
             <nextElementRef id="ProsumerCondition2" />
              <predicate>condition = 'true'</predicate>
          </flowsInto>
          <!- - Other elements connected to  ProsumerTask#1 - ->
          <join code="and" />
         <split code="or" />
         <decomposesTo id="xml_prosumer_task_one" />
     </task>
    <!- - Other task or condition declarations - ->
   </processControlElements>
</decomposition>
```

```
<decomposition id=" xml_prosumer_task_one " xsi:type=" NetFactsType ">
    <processControlElements>
      <inputCondition id="InputCondition_task_one">
          <!- - InputCondition description - ->
       </inputCondition>
       <task id=" AtomicProsumerTask1">
          <name> AtomicProsumerTask#1</name>
          <flowsInto>
             <nextElementRef id="OutputCondition_task_one" />
          </flowsInto>
          <!- - Split and join description  - ->
          <!- - Resourcing description  - ->
          <mapping>
              <expression query = prosumerService.one()/>
               <!- - Other mapping statements (auxiliary variables)  - ->
          </mapping>
          <decomposesTo id=" service1" />
     </task>
    <!- - OutputCondition declaration - ->
   </processControlElements>
</decomposition>
<!- - Other composite task decomposition (NetFactsType) - ->
<decomposition id=" service1" xsi:type="WebServiceGatewayFactsType">
     <!- - Auxiliary variables - ->
 </decomposition>
<!- - Other atomic task decomposition (WebServiceGatewayFactsType) - ->
<!- - XML  YAWL footer - ->
```

Figure 5: Prosumer process description

Once the process is completely described, it is translated into a new workflow description language by a *domain interpreter*. The domain interpreter generates a process description at prosumer level (including metadata) which depends neither on the technological domain nor on the underlying hardware platform. At this level, process is still described using prosumer services. Any executable language such as BPEL or IPM–PDL [58] can be used. In our case we are employing the XML-based version of YAWL. Figure 5 presents the output of the domain interpreter.

As can be seen in Figure 5, the obtained process description from the *domain interpreter* includes a secondary decomposition for each task in the process, then in those description files tasks are described as composite tasks (in YAWL terminology). However, in practice, these tasks remain atomic because, as can be seen, secondary decompositions only include one task, which is completely executed invoking one prosumer service. This "artificial" way of representing tasks is intended to simplify following steps in the process execution. In fact, in business execution, general tasks are not truly executed but delegated. Moreover, most delegation services in CPS cannot receive or execute a description file containing an isolated task, and it must be included in a complete workflow. As a solution, *domain interpreter* generates a workflow of composite tasks, where each task is composed by a workflow with only one task (see Figure 5 and Figure 6). Thus, in order to delegate a task, it is only necessary to send the corresponding decomposition (together with the metadata) to the target CPS.

In Figure 5, additionally, decompositions with the label *WebServiceGatewayFactsType* may be found. These statements are mandatory in YAWL in the case of invoking external services to the YAWL execution environment in order to execute tasks. In fact, in our case services are placed in the cloud, Internet or provided from an ad hoc network. They contain information about the input and output variables involved in the service execution. Nevertheless, this type of decompositions is not considered when analyzing whether tasks are atomic or composite in a certain description file.

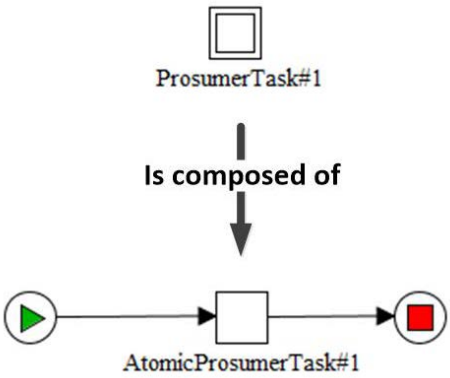


Figure 6: Prosumer process description

The file generated by the domain interpreter is stored in the *domain expert environment* until users decide to upload it to the execution system. The first element in the execution system is the first *semi-automatic transformation engine* which offers an asynchronous invocation API to the *domain expert environment*. Although many invocation APIs could be used (SOAP, for example), in this case we have selected a REST-based interface. Thus, users may perform four different actions. Table 2 describes those four possibilities.

Generally, the first *semi-automatic transformation engine* transforms the prosumer platform-independent process description generated by the domain interpreter into two different files: a business platform-independent process description and a metadata description file at business level. Process description language employed at business level may be the same employed at prosumer level or different. In our case we are maintaining YAWL as description language. At this point, processes are described using business services instead of prosumer services.

Table 2: Operation of the REST interface in the first transformation engine

Action (HTTP operation)	Required Input	Expected Output	Explanation
PUT	Process and metadata description document	Unique identification of the process	The process is uploaded to the execution system (transformed and stored in the business execution). It will be never executed if users do not activate the process.
POST	Process and metadata description document	Unique identification of the process and XML report about the execution results	The process is uploaded to the execution system (transformed and stored in the business execution) and executed (at the moment indicated in the metadata)
GET	Unique identification of the process	XML report about the execution results	The process corresponding to the provided identification is executed at the moment indicated in the metadata.
REMOVE	Unique identification of the process	Acknowledgement	The process corresponding to the provided identification is removed from the execution system (business engine)

In the literature, many different technologies to implement transformation engines may be found. Holistic methodologies such as the "Business Implementation Methodology" (BIM) [59] have been reported to be efficient and successful. The implementation of process model transformation engines is not the objective of this paper, but a brief description is provided. Detailed descriptions about these solutions are available in the state-of-the-art [60]. In fact, as part of the transformation engines, in order to translate processes from a very high abstraction level into executable code we provide a mapping between processes and software components. Based on MDA (Model Driven Architecture) principles we establish correlations between states in initial business process definition language and the states of the PIM (Platform Independent Model) and the PSM (Platform Specific Model) defined in MDA. A business process model at the PIM level, ready to be transformed into lower level process description language, is then established. The transformation process is completed by mapping platform-independent elements of the business process model into platform specific ones.

In respect to metadata, any of the available description languages may be used (EML, XML, ISO19506, etc.). However, two facts should be considered. First, prosumer users can include new and unforeseen metadata and, second, at business execution some tasks (and the corresponding metadata) might be delegated to external services (such as bank services) which have not agreed on a common document structure for describing metadata. Thus, in this proposal, we used the Resource Description Framework (RDF) in order to describe metadata. Figure 7 presents both description files for the process presented previously in Figure 4 and Figure 5.

As can be seen, the ProsumerTask#1 and ProsumerTask#3 tasks have been disaggregated into two workflows involving various additional tasks (i.e. additional services) in order to create the process description at business level. Besides, metadata information has been removed from the process description document. However, in general the structure of the description document is

the same. Tasks continue being described as composite tasks (indicated by the double square, see Figure 7(c)), and the only difference is the use of business services instead of the prosumer services. Differences could be more if different languages were employed.

Much more interesting, however, is the metadata RDF description document. As can be seen, firstly, the metadata related to the entire process are listed: start time, global quality or the number of times the process must be executed. Later, for each task at business level, some additional metadata are also included. In general, for example, quality parameters present (at this level) the structure cited in Section 3.1. In that way, task delegation becomes easier: it is only necessary to transmit to the target CPS the decomposition and metadata associated with the task.

```
<!- - XML  YAWL header - ->
<decomposition id="BusinessProcess" isRootNet="true" xsi:type="NetFactsType">
    <!- - Auxiliary variables declaration - ->
   <processControlElements>
      <!- - Other task or condition declarations - ->
      <task id="BusinessTask1a">
          <name>BusinessTask#1a</name>
          <!- - Other description statements - ->
          <decomposesTo id="xml_business_task_oneA" />
      </task>
      <!- - Other task or condition declarations - ->
   </processControlElements>
</decomposition>
<decomposition id=" xml_prosumer_task_one " xsi:type=" NetFactsType ">
  <processControlElements>
      <!- - InputCondition declaration - ->
      <task id=" AtomicBusinessTask1">
       <!- - Other description statements - ->
        <mapping>
            <expression query = businessService.oneA ()/>
            <!- - Other mapping statements (auxiliary variables)  - ->
        </mapping>
        <decomposesTo id=" service1" />
      </task>
      <!- - OutputCondition declaration - ->
  </processControlElements>
</decomposition>
<!- - Other composite task decomposition (NetFactsType) - ->
<!- - Atomic task decomposition (WebServiceGatewayFactsType) - ->
<!- - XML  YAWL footer - ->
```

```
<rdf:RDF
    <!- - RDF document header - ->
   <rdf:Description rdf:about="BusinessProcess">
      <quality:responseTime>12</ quality:responseTime >
      <quality:jitter>3</ quality:jitter >
      <!- - Other predicates about QoS - ->
      <temp:executionTimes>3</ temp:executionTimes >
      <!- - Other predicates about temporization - ->
      <!- - Other predicates about execution restrictions - ->
   </rdf:Description>
   <!- - Metadata description about other tasks - ->
  <rdf:Description rdf:about="BusinessTask#2">
      <!- - Other predicates about QoS - ->
      <temp:timer> HOUR_OF_DAY </ temp:timer >
      <!- - Other predicates about temporization - ->
      <exec:location> local </ exec:location >
      <!- - Other predicates about execution restrictions - ->
   </rdf:Description>
   <!- - Metadata description about other tasks - ->
</rdf:RDF>
```

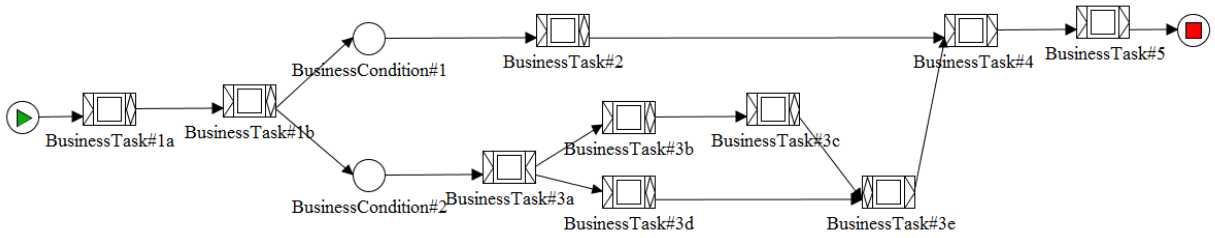


Figure 7: (a) XML business process description (b) Business metadata description (c) Graphic representation of the business process description

Although the first *semi-automatic transformation engine* offers four different services to prosumer users (see Table 2), it does not store, manage or execute any process. In fact, most of these functionalities are provided by the *business execution engine*. Table 3 describes the behavior of the transformation engine.

Table 3: Transformation engine behavior

Action (HTTP operation)	Explanation
PUT	The received process description is translated and sent to the business execution engine to be stored.
POST	The received process description is translated and sent to the business execution engine to be stored and executed if possible.
GET	Redirects the information to the business engine in order to initiate the execution of the corresponding process
REMOVE	Redirects the information to the business engine in order to remove the corresponding process

The transformation engine also generates the prosumer results reports (in this case using XML language) which are showed to users in the *result presentation* module. These reports are obtained by composing the business results reports by means of a kind of the "inverse algorithm" of the transformation algorithm. Figure 8 shows a possible results report about the process in Figure 7.

```
<results id="BusinessProcess">
    <localVariable>
      <name>variable1</name>
       <value>23</value>
    </localVariable>
    <!- - Other local variables - ->
    <task id = "ProsumerTask1">
      <complete>true</complete>
      <!- - Other reports - ->
    </task>
    <!- - Other task results - ->
</results>
```

Figure 8: Results reports at prosumer level

As final idea, many technologies can be used for implementing the transformation engine. Compilers technologies [5], sematic solutions [43] or specific technologies for descriptions refactoring [55] are valid.

3.2.2 Business execution

This second step concentrates most of the contributions of this work. Figure 9 shows a detailed functional architecture including the components involved in the business execution. In this step, business services are considered. These medium-level services are compositions of production services (see Section 3.2.3), focused on describing functionalities in a platform-

independent way, which helps tasks and processes to be delegated to external CPS. The corresponding repository, the *business services repository*, stores the catalogue of available services in the platform. It is the most important repository, and (besides the basic information about services) it must include (for each service) additional information about the execution cost and the guaranteed QoS. In particular, we are assuming this information is provided in the most typical way in CPS [33]: the cost is expressed as an integer, and the QoS is represented by a vector of integer values [52]defining each value a quality parameter (availability, jitter, etc.). Finally, two more remote repositories (not included in Figure 9) must be available: the repository of cloud services and the repository of Internet Services. These two repositories must present the same information of the business repository, but are managed by third-party entities, so they are not part of the proposed system. These repositories could be consulted through the *inter-system services API*.

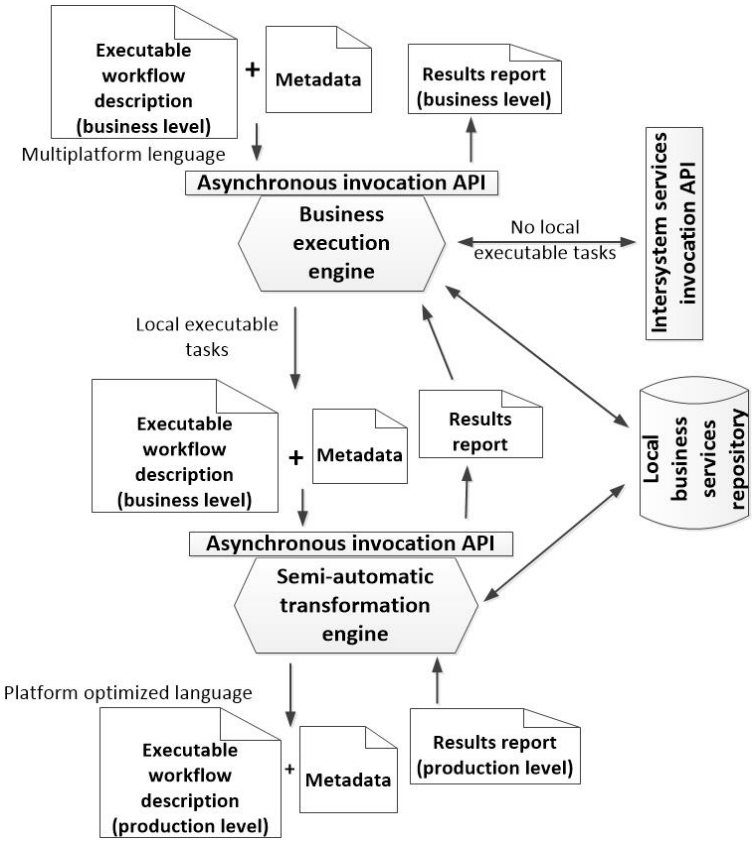


Figure 9: Detailed functional architecture for business execution

As we explained in Table 3, it is the *business execution engine* the entity which actually maintains and executes processes. This engine receives the output from the first semi-automatic transformation engine (the YAWL description at business level and the RDF document) and acts depending on the invoked HTTP operation and the information provided in the metadata. This engine presents towards the first transformation engine an asynchronous invocation API (REST-based in this case) whose behavior is exactly described in Table 2. In this case, all the described actions are in fact performed by the engine.

In order to perform all those functions, business execution engine is made up of various functional components. Figure 10 presents that structure.

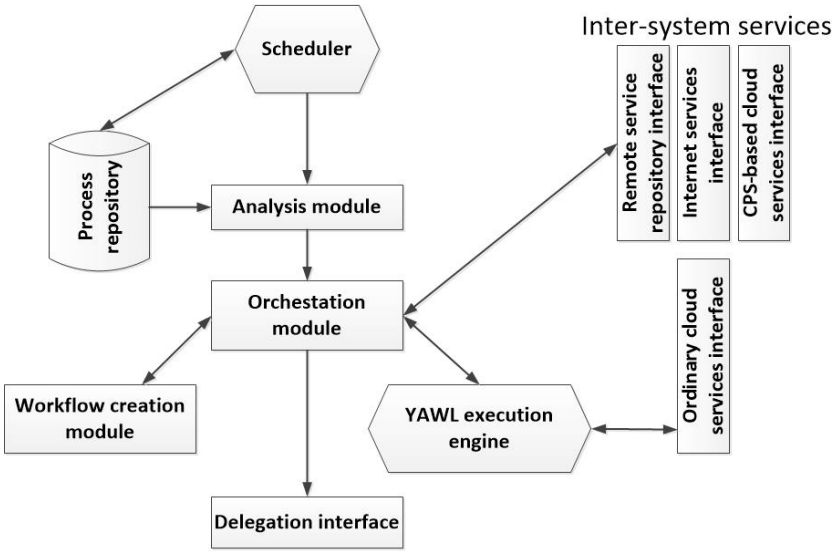


Figure 10: Functional architecture for the business execution engine

Operations invoked by users are received by the scheduler, which assigns the unique identification to the process and stores both documents (process and metadata description) in the process repository. If a GET or REMOVE operation is ordered, the corresponding actions are also performed. If a certain process gets active (because a GET or POST operation is executed), the scheduler loads the temporization metadata about the whole workflow. Then, when the start time conditions are reached the scheduler loads the process and metadata description documents into the analysis module.

The analysis module parses the process description document and locates the first-order business synchronization points. First-order business synchronization points are placed on the workflow knots (points where more than two elements are connected). The *InputCondition* and the *OutputCondition* are always also first-order business synchronization points. The sequence of elements (tasks and conditions) which connect two knots is a branch. One element is never shared by various branches. Figure 11 represents the branches and the first-order business synchronization points of process on Figure 7(c).

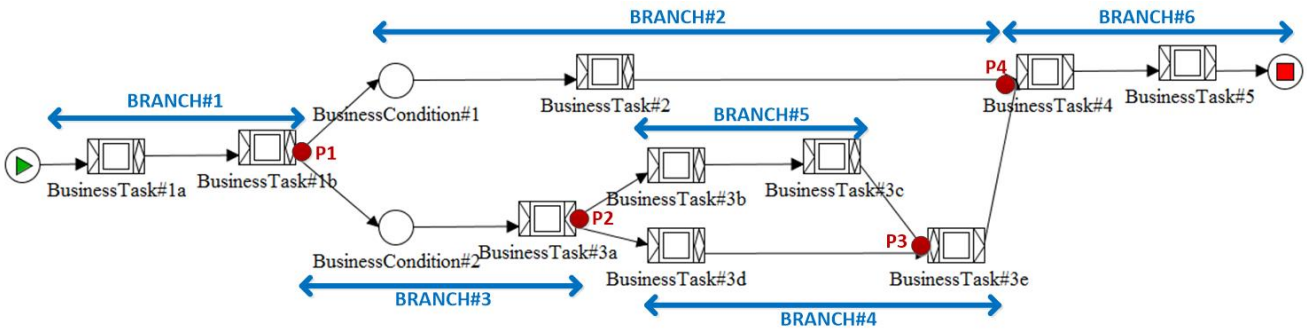


Figure 11: First-order business synchronization points and branches in a workflow

Considering the document structure showed in Figure 7(a), Algorithm 1 presents a solution to calculate the first-order business synchronization points of a workflow. At these points, at least, the execution will be stopped in order to evaluate the execution evolution. Moreover, in knots where various branches finish, the engine also synchronizes the different flows; waiting for some or all of them in OR-joins and AND-joins, or discarding all of them except the first in XOR-joins. It is important to note that Algorithm 1 includes references to XML fields that are only employed in YAWL workflow descriptions (see Figure 7(a)). Therefore, if other description language is employed, the names of these field should be replaced to which are used in the corresponding language.

Algorithm 1 Synchronization points calculation

Input: XML-YAWL process description document
Output: List of synchronization points *pointsList*
Create a new XML parser called *parser*
Invoke the parsing method of *parser* with the YAWL document
Create a new object XMLDocument *doc*
Assign *doc* to the XMLDocument object in *parser*
Create a list of nodes *nodeList*
Assign *nodeList* to a list containing all the elements under the tags "task" and "condition" in the document *doc*
Create a new list of nodes *pointsList*
for each *node* in *nodeList* **do**
  Obtain a list *childList* containing all the children of *node* with the tag "flowsInto"
  if *childList* contains at least two elements **then**
    Create a boolean variable *poinExists* equal to *false*
    for each *nodePoint* in *pointsList* **do**
      Obtain a list *childPointList* of all the children of *nodePoint* with the tag "joinElement"
      if *childPointList* and *childList* contain the same elements **then**
        Add to *nodePoint* a new child with the tag "splitElement" about the root element in *node*
        Assign *pointExists* to *true*
        Update *pointsList* with the new *nodePoint*
      end if
    end for
    if *pointExists* is *false* **then**
      Create a new node *nodePoint*
      Add to *nodePoint* a new child with the tag "splitElement" about the root element in *node*
      Add to *nodePoint* a collection of new children with the tag "joinElement" about the elements in *childList*
      Add *nodePoint* to *pointsList*
    end if
end for

When all first-order business synchronization points have been calculated, the workflow is divided into branches. Algorithm 2 describes a method to perform this action.

Once all first-order business synchronization points and branches on the workflow have been located, the analysis results and the process and metadata descriptions are transmitted to the orchestration module. The orchestration module executes the process branch-by-branch. However, if in some synchronization point various branches are born (typically in AND-splits or OR-splits where various conditions become true), then these branches are executed in

parallel. For example, in Figure 11 branches BRANCH#2 and BRANCH#3 (probably together with BRANCH#4 and BRANCH#5) will be executed in parallel.

Algorithm 2 Branches calculation

Input: XML-YAWL process description document and List of synchronization points *pointsList*
Output: List of synchronization branches *branchesList*
Create a new XML parser called *parser*
Invoke the parsing method of *parser* with the YAWL document
Create a new object XMLDocument *doc*
Assign *doc* to the XMLDocument object in *parser*
Create a new lists of nodes *branchesList*
Create a list of nodes *joinNodes* equal to all elements in *pointsList* with the tag "joinElement"
Assign the node *inputNode* to the node with tag "InputCondition" in *doc*
Obtain the list *childrenInput* of all children of *inputNode* with tag "flowInto"
for each *node* in *childrenInput* **do**
  Create a boolean variable *brancheComplete* equal to *false*
  Create a node *nextNode* equal to *node*
  Create a new node *branchNode*
  while *brancheComplete* is *false* **do**
    Add to *branchNode* the element *nextNode* with the same tag it presents in *doc*
    if *nextNode* is in *joinNodes* **then**
      Assign *brancheComplete* to *true*
      Add *branchNode* to the list *branchesList*
    else
      Obtain the child *nextChild* of *nextNode* in *doc* with the tag "flowInto"
      Assign *nextNode* to *nextChild*
    end if
  end while
end for
for each *nodePoint* in *pointsList* **do**
  Obtain a list *childPointList* of all the children of *nodePoint* with the tag "joinElement"
  for each *childPoint* in *childPointList* **do**
    Create a boolean variable *brancheComplete* equal to *false*
    Create a node *nextNode* equal to *childPoint*
    Create a new node *branchNode*
    while *brancheComplete* is *false* **do**
      Add to *branchNode* the element *nextNode* with the same tag it presents in *doc*
      if *nextNode* is in *joinNodes* **then**
        Assign *brancheComplete* to *true*
        Add *branchNode* to the list *branchesList*
      else
        Obtain the child *nextChild* of *nextNode* in *doc* with the tag "flowInto"
        Assign *nextNode* to *nextChild*
      end if
    end while
  end for
end for

The orchestration module may delegate the entire branch, execute it in the cloud or execute it locally. Additionally, second-order business synchronization points could be added and, then, each task or collection or tasks would be executed in a different location. In general, several of these options would be available. Then, a decision algorithm is required. This algorithm is based

on mandatory metadata (QoS and execution restrictions) about the tasks which compose the branch and information about the cost of executing the services in each available location. In particular, a hierarchy of requirements is proposed:

1. Execution restrictions indicated by users: If any task must be executed or cannot be executed in a certain location, all the possibilities including the forbidden options will be discarded.
2. Quality-of-Service: For each task (i.e. business service) only the locations which guarantee a higher quality than the one demanded by users will be considered.
3. Execution cost: Finally, if various possibilities remain adequate, the one which presents the lowest execution cost will be selected.

Execution restrictions indicated by users may be directly used as a first filter. For example, in Figure 7(b) it is indicated that BusinessTask#2 must be executed on the local ad hoc network. However, both other two criteria (QoS and execution cost) require a study about how to perform the selection.

As we said, information about the QoS is stored for each business service in the appropriate repository (local business service repository or remote repositories for cloud or Internet services). This information is presented as an n_k-dimensional vector of integer values (1), indicating the value of a collection of quality parameters (availability, response time, etc.).

$$QoS_{offer} = (\alpha_1, \alpha_2, \dots, \alpha_{n_k}) \qquad (1)$$

It must be noted that k parameter is an index to indicate the particular business service we are referring to. In general, the value of n parameter depends on the considered service (thus, for a particular one the value would be n_k). Moreover, the particular parameters selected for indicating the guaranteed QoS also depends on the service, location, etc.

On the other hand, from the metadata description document (RDF document) it is possible to deduct also a collection of quality parameters for each task (i.e. business service), as said in (2).

$$QoS_{request} = (\beta_1, \beta_2, \dots, \beta_{m_k}) \qquad (2)$$

As in the previous case, for each task the indicated quality parameters may not be the same, and the dimension of the vector $QoS_{request}$ can also vary (thus, for a particular one the value would be m_k).

In those conditions, a location L offers for a certain service k a guaranteed QoS higher than which required by users if the relation (3) is verified.

$$\sum_{i=1}^{m_k} \rho_i \beta_i \; < \; \sum_{j=1}^{n_k} \mu_j^L \alpha_j^L$$

Where ρ_i is the relative weight associated with the i-th quality parameter β_i included in the RDF document in relation to the service k. Besides, μ_j^L is the relative weight associated with the j-th quality parameter α_j^L, associated to the service k when executed in the location L. Relative weights are very important as they allow giving more relevance to some parameters than others. For example, to ensure that no service with a certain QoS parameter α_j^L below a hard limit

β_i indicated in the RFD document is invoked, the weight of this parameter must be more than 50% of the total combined QoS.

As an additional requirement, in order to obtain comparable results, it must be fulfilled that $\sum_{i=1}^{m_k} \rho_i = \sum_{j=1}^{n_k} \mu_j^L$.

Once all the available and adequate locations for each task have been obtained, various options in order to execute the branch are, in general, possible. It must be taken into account that for a branch including a total amount of e elements, the total possible execution schemes is 3^e (as three different locations are possible). Then, it is selected the execution scheme with the lowest cost.

Algorithm 3 Execution scheme selection

Input: List of elements (task or conditions) *elementList* which are included in the branch
Output: List of locations *locationsList*
Create a new list of URI *locationsList*
Create a new list of URI *locations = {local, Internet, cloud}*
Create a integer variable *numLocations* equal to the number of elements in *locations*
Create a new list of list of URI *possibleLocations*
Create three integers *i, j, k* and *counter* equal to zero
Create an integer *numElements* equal to the number of elements in *elementList*
Create a HashMap <List of locations, Integer> *map*
while *i* < *numElements* **do**
  counter and *j* are equal to zero
  while *j* < *numLocations* **do**
    Assign *k* to zero
    while *k* < $numLocations^i$ **do**
      Assign the *i*-th position of the *k*-th list in *possibleLocations* to the *j*-th element in *locations*
      Increment *k* and *counter* in one unit
    end while
    Increment *j* in one unit
    if *counter* < $numLocations^{numElements}$ and *j* >= *numLocations* **then**
      Assign *j* to zero
    end if
  end while
  Increment *i* in one unit
end while
for each *scheme* in *possibleLocations* **do**
  while *i* < *numElements* **do**
    Obtain the *i*-th element in *scheme*, and store the value in the variable *location*
    Read de metadata document and determine if *location* is a valid location for the *i*-th task in the branch
    Look for the service invoked in the *i*-th task in the service repository of *location*
    if *location* is not valid **or** the service is not available in *location* **then**
      Remove *scheme* from *possibleLocations*
    else
      Obtain the quality parameters $(\alpha_1, \alpha_2, \dots, \alpha_{n_k})$ about the *i*-th task in the repository of *location*
      Obtain the quality parameters $(\beta_1, \beta_2, \dots, \beta_{m_k})$ about the *i*-th task in the metadata document
      if $\sum_{r=1}^{m_k} \rho_r \beta_r > \sum_{j=1}^{n_k} \mu_j^L \alpha_j^L$ **then**
        Remove *scheme* from *possibleLocations*

```
            else
                Increment i in one unit
            end if
        end if
    end while
    if possibleLocations contains scheme then
        Create a URI variable oldLocation
        Create an integer variable numPoints
        while i < numElements do
            Obtain the i-th element in scheme, and store the value in the variable location
            Obtain the hardware cost of executing the i-th task in the branch in the corresponding
            repository of location and store the value in η_i^{L_i}
            if oldLocation is different from location then
                Increment numPoints in one unit
            end if
            Assign oldLocation to location
            Increment i in one unit
        end while
        Obtain the total cost of the scheme as Σ_{i=1}^{numElements} η_i^{L_i} + numPoints^2
        Add to map a new entry (scheme, total cost)
    end if
end for
Assign locationsList to the list of locations in map associated with the lowest cost
```

In general, the execution cost is obtained as the addition of two quantities. First, the hardware cost of executing a certain service k in a certain location L, η_k^L. And, second, the cost of the orchestration actions δ. This second quantity depends on the number of second-order synchronization points introduced, i.e. the number of fragments into which the entire branch is divided. Thus, the total execution cost for a certain execution scheme s is obtained as (4). Briefly, this expression adds the hardware cost of executing each service in the selected location L_i, up to a total of τ services that compose the branch under study, and later adds the orchestration cost, which is a function of the selected scheme.

$$c_s = \sum_{i=1}^{\tau} \eta_i^{L_i} + \delta_s = \sum_{i=1}^{\tau} \eta_i^{L_i} + f(s)$$

The values of $\eta_i^{L_i}$ are provided in the service repositories together with the quality information, and function $f(s)$ may be defined as desired. In our case, we are proposing a quadratic cost function (5). Where p_s is the number of second-order business synchronization points in the scheme s. As execution costs $\eta_i^{L_i}$, which are dimensionless values, usually normalized, $f(s)$ does not have any unit. It is only an indicator of the orchestration cost. Typical orchestration algorithms present a complexity of n^2 with the number of elements to manage. In our scenario, elements to manage are the parts in which a branch is divided, an amount that corresponds to the number of orchestration points. Therefore a quadratic function may be used to represent the computational cost of orchestrating a branch with $p_s$orchestration points.

$$f(s) = p_s^2 \qquad (5)$$

Finally, Algorithm 3 describes the decision algorithm which allows selecting the most adequate execution scheme. This algorithm calculates all the possible execution schemes, and, later,

discards every scheme which is forbidden by users, does not fulfill the quality requirements, or involves executing a service in a non-available location. In the last step, the scheme with the lowest execution cost is selected.

Mathematically, Algorithm 3 solves a graph problem (see Figure 12). Figure 12 represents the equivalent graph problem of calculating the execution scheme for a branch with three elements (tasks or conditions). The problem consists of finding the path between the initial node and the final node with the lowest cost. In expression (4), the total cost of moving among the nodes for the lowest cost path has been expressed as the cost function $f(s)$, however, if desired, it is also possible to associate a different cost for each path between each couple of nodes (see Figure 12) and substitute the cost function for the expression (6).

$$\sum_{i=1}^{\tau} \Psi_i^{L_i - L_{i+1}}$$

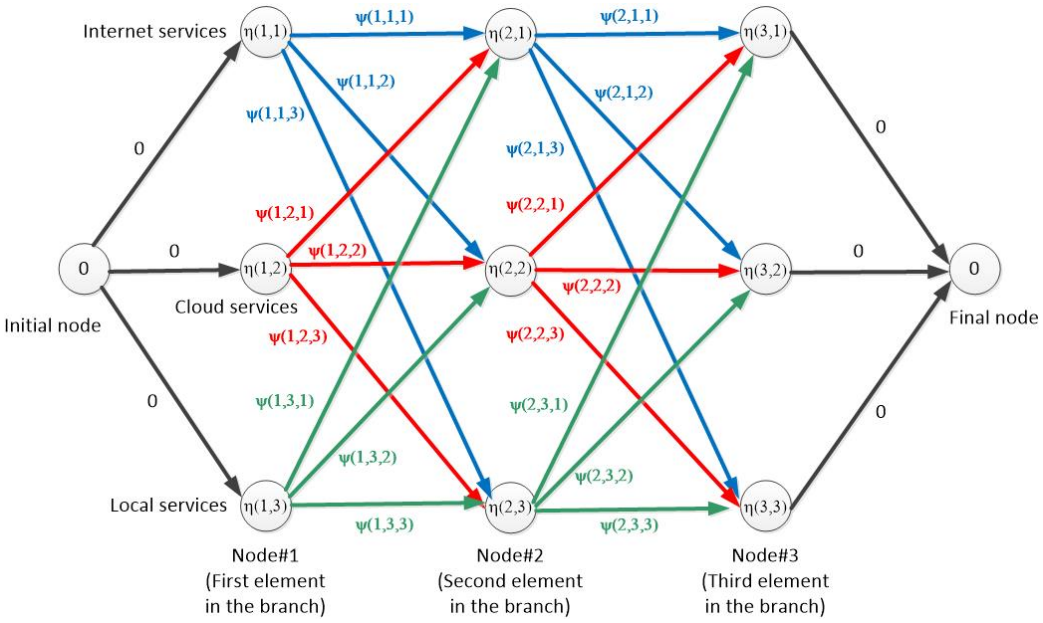


Figure 12: Equivalent graph problem for a branch of three elements. For representation purposes, subscripts have been reordered, consider that $\eta_i^{L_i} = \eta(i, L_i)$

When the most adequate execution scheme for each branch is selected, then, information about the elements (tasks and conditions) which compose the branch is extracted from the process description document and from the metadata description document. At this point, RDF technology and the use of composite tasks in the YAWL description simplifies the processing activities.

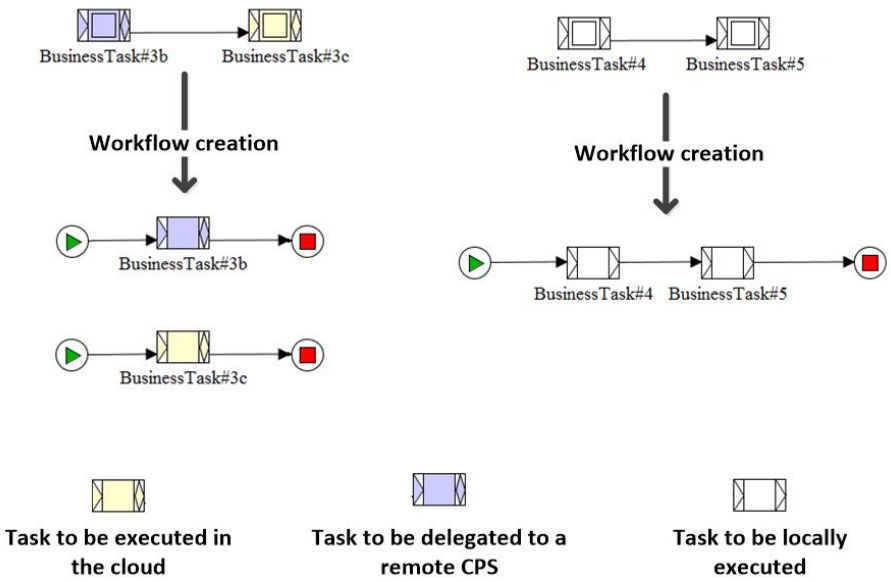


Figure 13: Workflow creation process in the business engine

This information is transmitted to the *workflow creation module*, which elaborates a new (or various) complete process description and metadata description documents related to the branch to be executed. In general, it creates a new workflow at business level with the tasks between every couple of second-order business synchronization points (see Figure 13). The output of this module is a collection of YAWL descriptions (using now atomic tasks) and a collection of metadata descriptions which can be finally delegated or executed (see Figure 14). As a novelty, the module could add some additional metadata, for example, about the expected precision in the probabilistic execution. Moreover, information about QoS and execution restrictions can be removed.

```
<!- - XML  YAWL header - ->
<decomposition id="BusinessProcessBRANCH#6" isRootNet="true" xsi:type="NetFactsType">
      <!- - Auxiliary variables declaration - ->
    <processControlElements>
       <!- - Other task or condition declarations - ->
       <task id="BusinessTask4">
           <name>BusinessTask#4</name>
          <!- - Other description statements - ->
          <mapping>
              <expression query = businessService.four ()/>
              <!- - Other mapping statements (auxiliary variables)  - ->
          </mapping>
          <decomposesTo id=" service4" />
       </task>
       <!- - Other task or condition declarations - ->
    </processControlElements>
</decomposition>
<decomposition id=" service4" xsi:type="WebServiceGatewayFactsType">
      <!- - Auxiliary variables - ->
 </decomposition>
<!- - Other task decomposition (WebServiceGatewayFactsType) - ->
<!- - XML  YAWL footer - ->
```

```
<rdf:RDF
    <!- - RDF document header - ->
   <rdf:Description rdf:about="BusinessProcesBRANCH#6s">
       <temp:startTime>HOUR_OF_DAY </ temp:startTime >
       <!- - Other predicates about temporization - ->
       <probab:precision>0.01</ probab:precision >
       <!- - Other predicates about the probabilistic execution - ->
   </rdf:Description>
  <!- - Metadata description about other tasks - ->
  <rdf:Description rdf:about="BusinessTask#4">
       <temp:startTime>HOUR_OF_DAY </ temp:startTime >
       <!- - Other predicates about temporization - ->
       <probab:statFunc>Medium</ probab:statFunc >
       <!- - Other predicates about the probabilistic execution - ->
   </rdf:Description>
  <!- - Metadata description about other tasks - ->
</rdf:RDF>
```

Figure 14: Output of the workflow creation module for the BRANCH#6 of Figure 11

Four different actions can be taken by the business engine at this point. If a workflow must be executed locally, the corresponding process and metadata description is sent through the *delegation interface*. In another case, the workflow must be delegated to a remote CPS (using the appropriate interface). More complex is the execution of services in the cloud. Sometimes, cloud services are CPS-oriented, and then the process description and metadata description may be sent through the appropriate interface as the previous cases. However, other times, cloud services are not CPS-oriented, and the process and metadata description cannot be directly sent. In those cases, process description is sent to an internal YAWL engine (connected to the cloud) which executes the process at business level and requests the cloud services using ordinary techniques [46].

Tasks which are delegated to cloud or Internet services will be executed in those systems and only results report will return. In the case of tasks to be executed locally, the execution continues.

As showed in Figure 9, process descriptions at business level for tasks to be executed locally are translated into platform-dependent language by a second *semi-automatic transformation engine*. This engine presents a behavior identical to the first transformation engine, including the same interface explained in Table 2 and Table 3. In this transformation process, platform-independent descriptions are transformed in platform-optimized descriptions (see Figure 15). Basically, although a second language could be used if necessary, business services are replaced by production services. In our case, we are maintaining YAWL as description language.

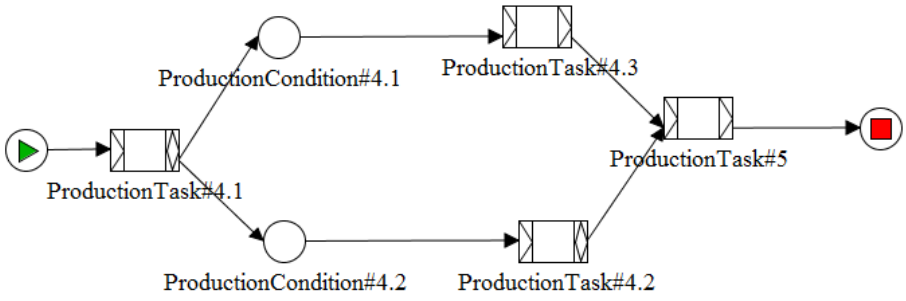


Figure 15: Workflow at production level for the BRANCH#6 on Figure 11

In respect to metadata, they are also transformed into production level. At this level, all metadata must be referred to a particular characteristic of the underlying ad hoc network. For example, if the ad hoc network contains temperature sensors, the maximum admissible error in temperature should be added as metadata. This kind of information will be very important for the probabilistic execution.

In this first work, three metadata about probabilistic execution are considered. First, the maximum admitted error in measures. Second, the number of realizations required for each measure. And third, the statistic function that is applied to the realizations in order to obtain the final value (medium, median, etc.). Figure 16 represents the process description and the metadata document obtained as result of this second transformation process.

```
<!- - XML  YAWL header - ->
<decomposition id="ProductionProcess" isRootNet="true" xsi:type="NetFactsType">
      <!- - Auxiliary variables declaration - ->
    <processControlElements>
       <!- - Other task or condition declarations - ->
       <task id="ProductionTask41">
            <name>BusinessTask#4.1</name>
           <!- - Other description statements - ->
           <mapping>
               <expression query = productionService.four1 ()/>
               <!- - Other mapping statements (auxiliary variables)  - ->
           </mapping>
           <decomposesTo id=" service41" />
       </task>
       <!- - Other task or condition declarations - ->
    </processControlElements>
</decomposition>
<decomposition id=" service41" xsi:type="WebServiceGatewayFactsType">
      <!- - Auxiliary variables - ->
 </decomposition>
<!- - Other task decomposition (WebServiceGatewayFactsType) - ->
<!- - XML  YAWL footer - ->
```

```
<rdf:RDF
    <!- - RDF document header - ->
   <rdf:Description rdf:about="ProductionProcess">
      <temp:startTime>HOUR_OF_DAY </ temp:startTime >
      <!- - Other predicates about temporization - ->
      <probab:maxError>10%</ probab:maxError >
      <!- - Other predicates about the probabilistic execution - ->
   </rdf:Description>
  <!- - Metadata description about other tasks - ->
  <rdf:Description rdf:about="ProductionTask#4.1">
      <temp:startTime>HOUR_OF_DAY </ temp:startTime >
      <!- - Other predicates about temporization - ->
      <probab:numRealizations>5</ probab: numRealizations >
      <probab:statFunc>Medium</ probab:statFunc >
      <!- - Other predicates about the probabilistic execution - ->
   </rdf:Description>
  <!- - Metadata description about other tasks - ->
</rdf:RDF>
```

Figure 16: Process and metadata description at production level

The second transformation engine also generates, for each task or branch executed in the local ad hoc network, the corresponding results report at business level (composed from the reports at production level). These reports are received by the business engine which must combine them with the ones received from the cloud of remote CPS to build the global report at business level.

3.2.3 Probabilistic execution

In the probabilistic execution step, the workflow is finally executed. Figure 17 offers a detailed functional architecture of the components involved in the probabilistic execution. This layer must absorb all the particular characteristics and aleatory behaviors of the hardware platform, and offer a “hard execution” to the upper components, i.e. the same workflow executed in two different instants offers the same results. In this step, tasks are described using production services. Production services can be directly translated into executable orders (in the format required by the underlying ad hoc network), but they are platform-dependent low-level services. In this case, the *production services repository* must include the same information as the business repository (see Section 3.2.2), besides the location of each service if it is provided by physical or virtual devices. Apart from this mandatory information, any other additional data may be stored.

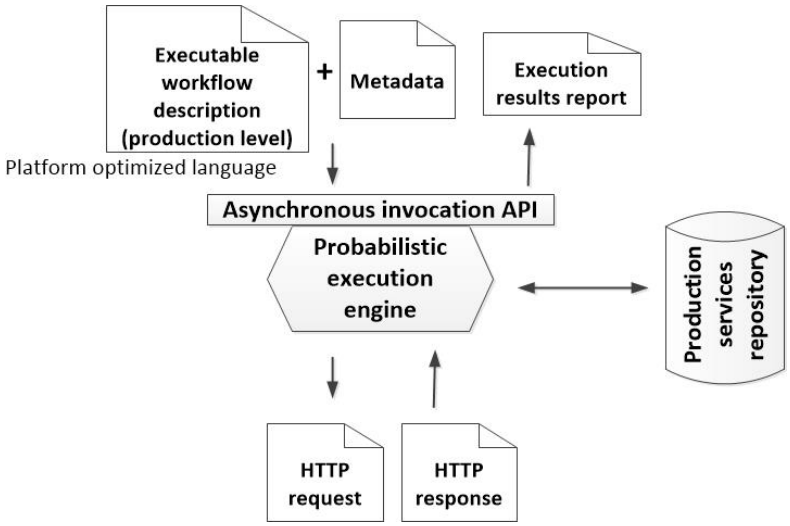


Figure 17: Functional architecture for the probabilistic execution

The *probabilistic engine*, basically, receives the process description document at production level and the metadata document and executes every task or condition, considering the provided metadata. Executing a task or condition implies the invocation of a certain service, which in our case, is offered by a hardware access infrastructure. These services are REST-based services, so the *probabilistic engine* must generate the corresponding HTTP requests (in general GET operations) and, as a result, will receive the resulting HTTP response (see Figure 18)

```
GET  productionService.four1

User-Agent:  ProbabilisticEngine
Host:  localNetwork
Accept-language: en-us
Connection: Keep-alive
```

```
HTTP/1.1 200 OK

Date: Mon, 12 Jul 2012  12:42:33 GMT
Server: localNetwork
Content-Length: 88
Content-Type: text/xml
Connection: Closed

<xml>
  <result>32.5</result>
</xml>
```

Figure 18: Example of the generated HTTP request and response by the probabilistic engine for the productionTask#4.1

Some additional components can be distinguished as part of the probabilistic engine (see Figure 19).

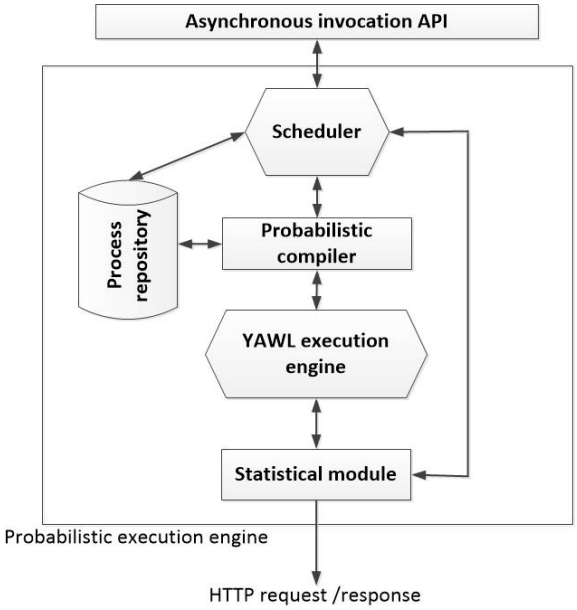


Figure 19: Functional architecture of the probabilistic execution engine

As seen in Figure 19, and similar to the case of the business engine, the probabilistic engine also implements a *scheduler* which receives REST operations. This scheduler assigns a unique identification to any delegated process and stores it in the *process repository*. The assigned identifications are sent back to the orchestration module in the business engine which maintains them until a user invokes a REMOVE operation at prosumer level. When a process gets active (because a GET or POST operation is received), the *scheduler* loads the metadata description document and evaluates the temporization conditions. When these conditions are fulfilled, it reads the process description document from the repository and sends it to a *probabilistic compiler*. That compiler also receives from the *scheduler* the metadata information about the probabilistic execution. In particular, it receives information about the maximum admissible error in the variables' value.

Table 4: Relation between hard and probabilistic control flow conditions

Maximum admissible error: ε_{max}	
Hard control flow condition	**Probabilistic control flow condition**
$var > \alpha$	$var > \alpha + \varepsilon_{max}$
$var < \alpha$	$var < \alpha - \varepsilon_{max}$
$var = \alpha$	$var \geq \alpha - \varepsilon_{max}$ && $var \leq \alpha + \varepsilon_{max}$
$var \geq \alpha$	$var \geq \alpha + \varepsilon_{max}$
$var \leq \alpha$	$var \leq \alpha - \varepsilon_{max}$

This compiler transforms the hard control flow conditions (i.e. the relations of equality or order which involves variables whose value depends on a service execution), into new conditions considering the maximum admissible error indicated in the RDF document. Table 4 presents the main hard control conditions which are present in a process description and the resulting probabilistic control conditions obtained from the *probabilistic compiler* for each case.

The obtained process description from the *probabilistic compiler* is sent to an internal YAWL engine which executes the process. The YAWL engine executes the workflow, and invokes the services composing every task and condition. The behavior of this engine is the standard [46].

At the same time, the *scheduler* reads the metadata document and extracts the remaining information about the probabilistic execution. This information is loaded in a *statistical module* which intercepts the service invocations (HTTP requests) done by the YAWL engine. This interception scheme allows employing standard stable implementations for the YAWL execution engine, which reduces the costs and complexity of the system. Two different types of services may be invoked: data collection services and actuation services. Depending on the type of service involved in the intercepted HTTP request, the *statistical module* runs different algorithms. In order to distinguish the service type, it may be indicated in the metadata (although it is not mandatory in our proposal) or may be deducted from the number and type of output and input variables. Typically, data collection services ask for different output variables (integer, string, etc.), and action services provide input variables, although sometimes *boolean* output variables are also considered.

Data collection services are intended to obtain certain data from the ad hoc network. If one of these services is invoked, the *statistical module* analyzes the corresponding metadata and creates a probabilistic study based on each intercepted HTTP request and the metadata information. This study basically consists of repeating the operation the number of times indicated in the metadata (or the required number until the hardware error is below the maximum indicated in the metadata), and later obtaining from the results the final value applying the indicated statistical function. For example, considering the information provided in Figure 15, when the YAWL engine executes the task productionTask#4.1 and invokes the service productionService.four1, the *statistical module* intercepts the HTTP request and performs a probabilistic study based on five different evaluations of the service productionService.four1. The medium of the obtained values will be returned as final result to the YAWL engine in the HTTP response by the *statistical module*. These probabilistic studies pretend to wean the computational processes from the aleatory facts typical of hardware devices and physical world. Algorithm 4 describes the behavior of this statistical module.

Algorithm 4 Statistical module behavior for data collection services

Input: HTTP request, RDF document about the task under execution
Output: HTTP response to the YAWL engine
Create a HashMap <String, function pointer> *functions* containing all the statistical functions
Create a new RDF parser called *parser*
Invoke the parsing method of *parser* with the RDF document
Create a new object RDFDocument *doc*
Assign *doc* to the RDFDocument object in *parser*
Create an integer variable *numRealizations*
Obtain the value of the property "probab:numRealizations" in *doc* and store it in *numRealization*
Create a list of execution results *resultsList*
Create an integer variable *counter* equal to *numRealizations*
while *counter > 0* **do**
  Send the HTTP request
  Wait for the HTTP response
  if HTTP response present a 200-OK code **then**
    Extract the execution result from the XML document
    Store the execution result in *resultsList*
  else
    Send the HTTP response to the YAWL engine
    End the algorithm
  end if
  Decrement *counter* in one unit
end while
Obtain the value of the property "probab:statFunc" in *doc* and store it in a new variable *key*
Obtain in *functions* the pointer associated with *key* and store it in the new variable *statisticalFunction*
Invoke *statisticalFunction* with *resultsList* and store the result in a new variable *finalResult*
Send an HTTP response with code 200-OK indicated the value of *finalResult* to the YAWL engine

On the other hand, actuation services are focused on performing a certain action,e.g. moving robotic arm, turning on the heater, etc. In this case the *statistical module* only performs two actions. First, input variables are rounded in order to adjust the precision of these values to which provided by the ad hoc network and to which applied by users in the metadata. Second, if finally any input variable presents a value smaller than the global precision, then this variable is removed from the HTTP request before being sent. If it is the only variable in the request, the corresponding HTTP request is not transmitted and the *statistical module* directly creates a 200-OK response to be sent to the execution engine. For example, considering a robotic arm which its movement precision was one degree, every robotic arm's input variable would be rounded to an integer value. Moreover, HTTP requests applying for movements smaller than one degree would be not transmitted.

Additionally, the *statistical module* may generate a report about the "quality" of the execution; i.e. about the final error committed ε_t which depends on the hardware error ε_h, experimental error ε_e and mathematical error ε_m in the ad hoc network and the precision applied by users ε_u, as well as the statistical function applied f_s. See expression (7). In some occasions, the precision applied by users ε_u is given in the form of the number of realizations which must be performed n.

$$\varepsilon_t = F\left(\varepsilon_h, \varepsilon_e, \varepsilon_m, \frac{\varepsilon_u}{n}, f_s[\cdot]\right)$$

These reports are directly transmitted to the *scheduler*, which can include this information in the production results reports. Then, this information would be transmitted to the business and the prosumer level in order to inform users about the obtained final precision.

3.2.4 Hardware access

In the last step, service invocations (HTTP requests) from the probabilistic execution engine (particularly from the statistical module) must be mapped into execution orders adequate for the underlying ad hoc network. Moreover, virtual devices can be included, so this mapping process must consider also this possibility. Figure 20 shows a functional architecture of the proposed solution for this final step.

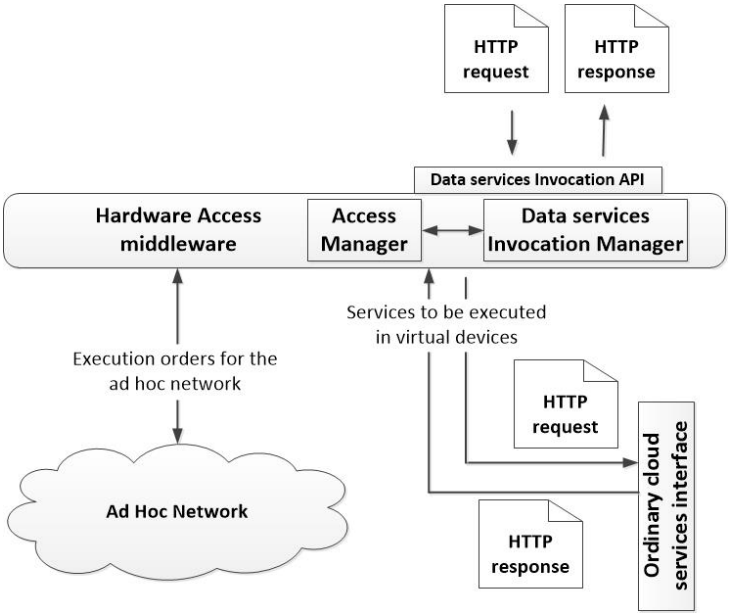


Figure 20: Functional architecture for the hardware access

As can be seen, the proposed solution is based on a middleware which connects the probabilistic execution engine with the ad hoc network and, if necessary, the cloud. The middleware offers an interface to the probabilistic engine based on REST services (as in the other cases). Once a HTTP request is received, the *Access manager* decides if that service is provided by the underlying ad hoc network or if it is supported by virtual devices and, then, it must be redirected to the cloud.

If the invoked service is supported by virtual devices, then the *Access manager* sends the original HTTP request to the *Data Services invocation manager*, which creates a new HTTP request indicating the web location and the name of the cloud service implementing the virtual device. This request is sent through the appropriate interface. When the HTTP request is received, the *Data Services invocation manager* builds a new response adequate for the original request which is sent to the probabilistic engine by the *Access Manager*.

On the other hand, if the invoked service is provided by the local ad hoc network, the *Access manager* only indicates to the *Data Services invocation manager* the service to be executed (the original HTTP request is not sent). Then, the *Data Services invocation manager* creates the appropriate execution orders and asks the ad hoc network to perform those actions. The result is sent back to the *Access manager* which creates the HTTP response.

Execution order may follow many formats, including Publication/Subscription messages [7], radio signals which are collaboratively addressed [57] or frames from the mobile technologies [54].

3.3 OTHER IMPORTANT ASPECTS: SECURITY, SCALABILITY AND FAULT RESILIENCE

The proposed solution and framework presents clear advantages for future industrial companies (those will appear after the fourth industrial evolutions), new smart buildings, and modern intelligent management solutions for critical infrastructures (among other applications). The described technology allows sharing a common production infrastructure among different companies or users, so costs are drastically reduced. New paradigms such as Manufacturing as a Service (MaaS) [61], Drone as a Service (DaaS) [62] or Sensing as a Service (SaaS) [63] could be easily supported by means of the proposed techniques.

However, the intrinsic complexity of CPS, together with difficulties associated to process management and transformation, cause the proposal to present some problems. First, to be effective, many heavy software components must be deployed and connected. Transforming and executing processes are costly operation, so the proposed solution is not valid for constraint-resources systems (for example, isolated low-power wireless sensor networks). Besides, although it is not an essential requirement, a high-quality communication infrastructure is needed. The efficiency of the proposed scheme depends strongly on the capacity of components to interact pretty fluently. Opportunistic communications and other similar technologies reduce in a very dramatic manner the quality of services supported by our proposal. Furthermore, as many interconnected services are required to maintain the global system available, these systems are vulnerable to Denial of Service (DoS) attacks.

Any case, different characteristics of the system guarantee its security and protection and its fault resilience. In respect to the first topic (security), REST interfaces may be easily extended to implement secure protocols, usually based on Public Key Infrastructures and certificates. In particular, HTTP protocol can be replaced by HTTPS (Secure HTTP) mechanisms. Besides, in order to avoid illegal accesses, a token-based solution can be deployed in combination to REST services. Most recent engineered REST services, such social networks or Openstack [64], employ this technology [65]. In relation to the second topic (fault resilience), the proposed scheme tolerates a huge amount of types of faults. Each REST request may be answered either with a success message or with a fault notification. Task, activity, branch or process may be paused by the execution engine in the higher level, and be resumed later. If the fault remains, and depending on the process definition and the selected configuration, the execution can continue or the user can be notified using the domain expert environment.

On the other hand, because of the complexity of the proposed architecture, scalability is probable one of the key issues to be faced. In fact, some important considerations must be done in relation to the maximum number of processes being executed at the same time. The

maximum number of processes to be executed at the same time depends strongly on the selected implementation of the YAWL execution engine and the underlying hardware platform. No studies have been reported about the scalability of most common implementation, based on Java technologies. Nevertheless various scalable implementations of YAWL engines have been proposed [66][67]. Furthermore, the use of REST technologies, instead of SOAP for example, guarantees that scalability is only conditioned by execution engines and not by the communication components.

Finally, it is important to note that users may configure the system to accept different definitions of costs, metrics, parameters… The domain expert environment (using the domain interpreter) and the first transformation engine should only include the homogenization laws to transform user definitions into "user restrictions" according to the defined data formats. In that way, parameters can be defined in a heterogeneous manner, according to user abilities or knowledge.

4. EXPERIMENTAL VALIDATION

An experimental validation was carried out in order to evaluate the performance of the proposed solution.

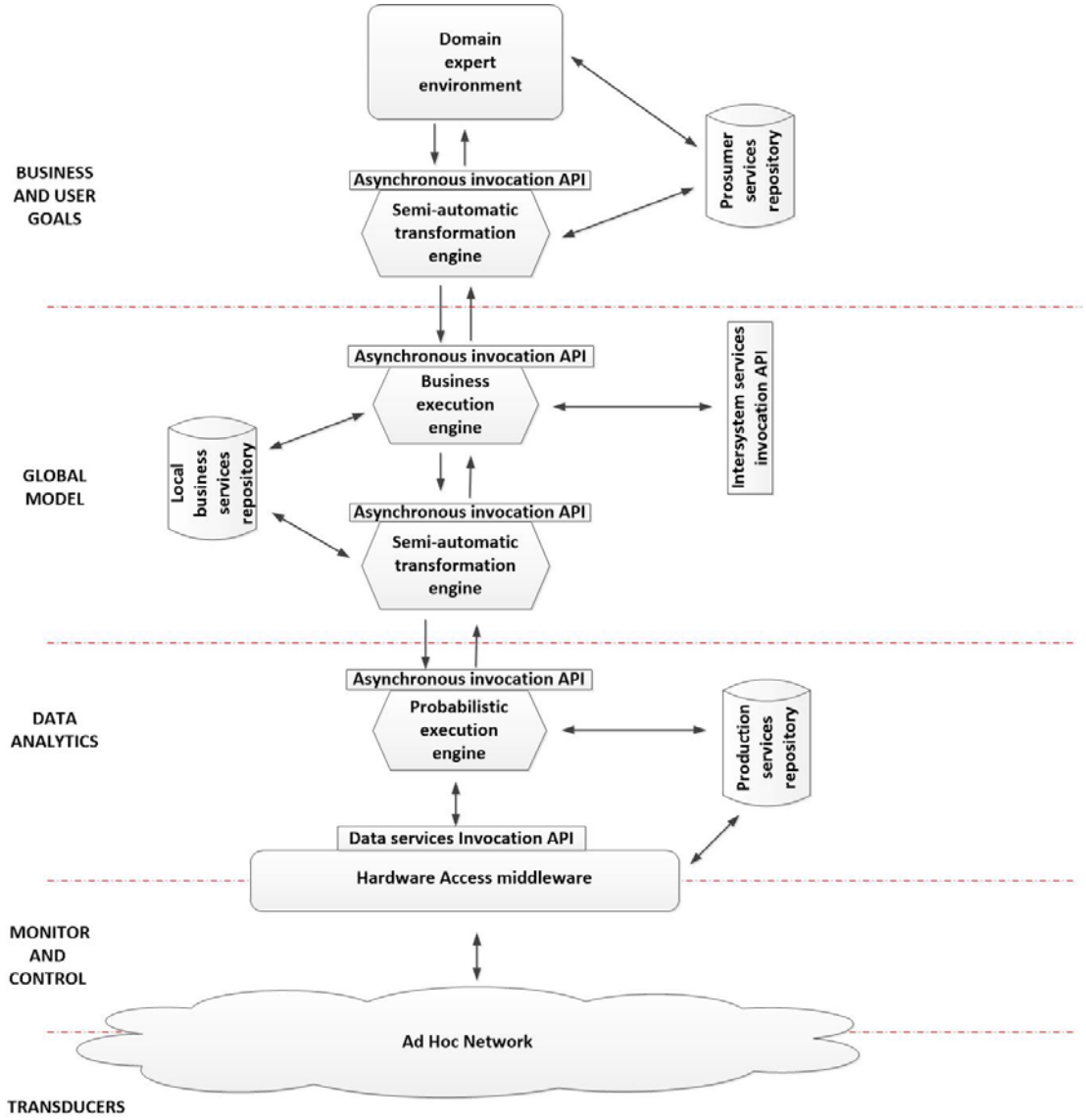


Figure 21: Complete functional architecture for the experimental validation

The validation consisted of one experiment where several processes tried to be executed. The validation was performed using a real deployment of the proposed solution, where simulated components and real software components were connected with real ad hoc network. Every component in the CPS created a log file describing its operation during the experiment. Finally, from the analysis of this log files, various results and conclusions were extracted.

Functional architectures represented in Figure 3, Figure 9, Figure 10, Figure 19 and Figure 20 were connected in order to create the complete process execution system for CPS (see Figure 21). All these components were implemented using Java technologies in a Linux server. In particular, databases were implemented using HSQLDB solutions [40], interfaces were based on RESTful library [36] and YAWL engines were implemented using the public distribution in Java [46]. Additionally, all described algorithms were also implemented using the Java programming language.

On the other hand, a real ad hoc network including sensor and actuator nodes was deployed. Each sensor node in the ad hoc network included a temperature sensor, a humidity sensor, a presence sensor and a light intensity sensor. Each actuator node included a buzzer and a light

bulb. The complete network consisted of two actuator nodes and five sensor nodes. Nodes in the ad hoc network communicated over ZigBee [39] wireless technology. Over this radio communications technology, a MQTT publication/subscription (P/S) system was deployed in order to transmit the execution orders to the nodes (and the results from the nodes to the P/S broker). Some previous works have showed the details of this configuration [29].

In order to connect the ad hoc network with the Linux server where software components were deployed, the server was provided with a ZigBee antenna. Besides, the P/S broker was also deployed in the Linux server and communicated through an Internet socket with the *Data services invocation manager*. In that way, the ad hoc network gets connected to the process execution system (in order to obtain the complete CPS), by means of a Hardware Access Middleware based on the P/S paradigm. Figure 22 shows all components involved in the described implementation.

Figure 22: Deployment for the experimental validation

The described system was deployed at the Technical University of Madrid and represents a basic CPS-supported Smart Home, one of the most relevant applications for CPS and process execution systems where users do not have technological skills [4].

The proposed CPS acted as local system. In order to validate all functionalities of the proposed solution, this system should be connected to other CPS through the Cyber-Physical Internet, and be connected to the cloud. However, in order to simplify the deployment, both locations were simulated using Linux virtual machines hosted in the Linux server. In total, four virtual machines were implemented: three of them ran software agents simulating remote CPS and the fourth machine ran a virtual cloud. All machines were created using Kernel-based Virtual Machines (KVM) [20] by means of a python script and the Libvirt API [17]. The connection among the virtual machines and the Linux server was made through the public Internet using public IP addresses.

Table 5: Available services in the local CPS for the experimental validation

Level	Service ID	Name	Description
Production	Production#1	Production.getTemp	It returns the value of the last temperature measurement done by the specified node.
	Production#2	Production.getHum	It returns the value of the last humidity measurement done by the specified node.
	Production#3	Production.getLight	It returns the value of the last measurement of the light intensity done by the specified node.
	Production#4	Production.getPresen	It returns a *boolean* indicating if the presence sensor is activated or not
	Production#5	Production.setLED	It changes the state of the light bulb in the specified actuator node
	Production#6	Production.setAlarm	It changes the state of the buzzer in the specified actuator node
Business	Business#1	Business.getFireInf	It obtains the data related to the possible presence of a fire from the node or nodes placed in an specified area
	Business#2	Business.getPeopleInf	It obtains the data related to the people state from the node or nodes in an specified area
	Business#3	Business.alarm	It activates an alarm indicating a dangerous situation
	Business#4	Business.systemReset	It returns all the system to the initial state (no alarm)
Prosumer	Prosumer#1	Prosumer.isFire	It returns a *boolean* indicating if there is a fire in the indicated area
	Prosumer#2	Prosumer.getDangerLevel	It returns an integer indicating the danger level in the indicated area
	Prosumer#3	Prosumer.activateEvac	It activates the execution of the evacuation plan
	Prosumer#4	Prosumer.reset	It returns the system to the initial state (no evacuation)

In order to reduce the influence of external variables in the results, the system was manually configured. In particular, service descriptions in the service repositories were stablished manually and in a fixed way for all the experiment's duration. Table 5 describes the considered services for this validation. As can be seen, services are typical of an emergency alert system. Information about the execution costs and the guaranteed QoS was randomly configured.

Additionally, remote CPS and the cloud also offer some services at business level. Table 6 describes these services.

Table 6: Available services in the remote locations for the experimental validation

Location	Service ID	Name	Description
Cloud	Cloud#1	Cloud.calDangerLevel	It determines the danger level in an area whose ambient parameters are sent
Cloud	Cloud#2	Could.calFire	It determines if a fire is active in an area whose ambient parameters are sent
CP Internet	Internet#1	Internet.setSprinkler	It activates the sprinkler system of the building
CP Internet	Internet#2	Internet.closeDoor	It activates the automatic door closure system of the building
CP Internet	Internet#3	Internet.setOutAlarm	It activates the exterior horn associated with the building

As we said in Section 3, prosumer services are compositions of business services; and business services are compositions of production services. Then, services in Table 5 and Table 6 are interrelated. These decompositions are also manually stablished in a fixed way. Thus, no process composition technology must be deployed or included in the experimental validation (which reduces the variables to be taken into account). Table 7 shows these decompositions.

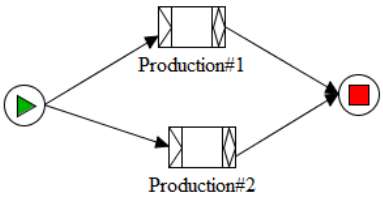
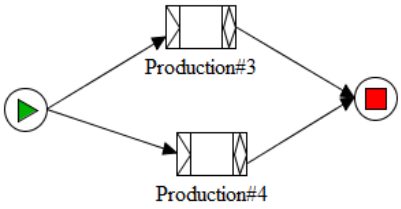
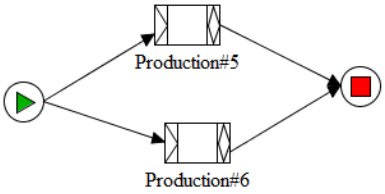
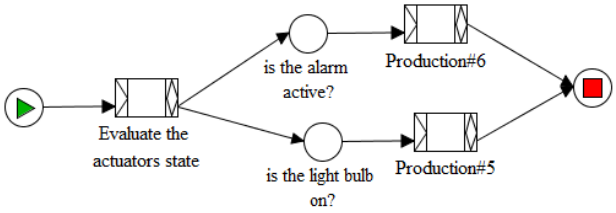
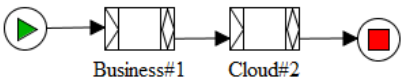
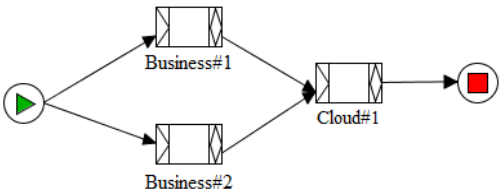
Considering the previously described infrastructure, the following experiment was performed. Every component in the local execution system was programmed to create a log file. Besides, a script to randomly generate processes at prosumer level was created. Sixty (60) different processes were obtained. During the experiment each one of these processes was executed four times. In total two hundred and forty (240) registries were obtained.

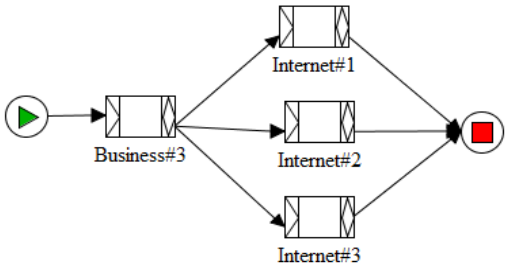
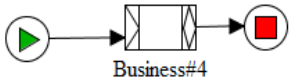
At the end of the experiment, all log files were collected and analyzed. In order to validate the proposed solution a process is considered to be correctly executed if various conditions are verified:

1. The ad hoc network executes the desired actions indicated by users at prosumer level, and follows the desired parameters (QoS, maximum admissible error, etc.)
2. The algorithm based on studying the process at business level branch by branch finally obtains the lowest cost execution scheme.
3. Experimental errors do not cause the appearance of false results.

If any of the three cited conditions is not verified, the execution process is considered incorrect, although a valid result is obtained.

Table 7: Service decompositions for the experimental validation

Level	Service ID	Decomposition
Business	Business#1	
		Description: parallel branch
	Business#2	
		Description: parallel branch
	Business#3	
		Description: parallel branch
	Business#4	
		Description: parallel branch and branch caused by condition
Prosumer	Prosumer#1	
		Description: sequential execution
	Prosumer#2	
		Description: parallel branch and sequential execution

	Prosumer#3	
		Description: sequential execution and parallel branch
	Prosumer#4	
		Description: sequential execution

5. RESULTS

From the analysis of the log files generated by every component in the deployed CPS, obtained during the described experiment (see Section 4), various results can be extracted.

First Figure 23 shows the percentage of processes successfully executed. As can be seen 95.4% of the processes were successfully executed (the three proposed conditions were fulfilled). Additionally, around in a 4.6% of cases the system was not able to correctly execute the process designed by users. The causes of these fails are studied below.

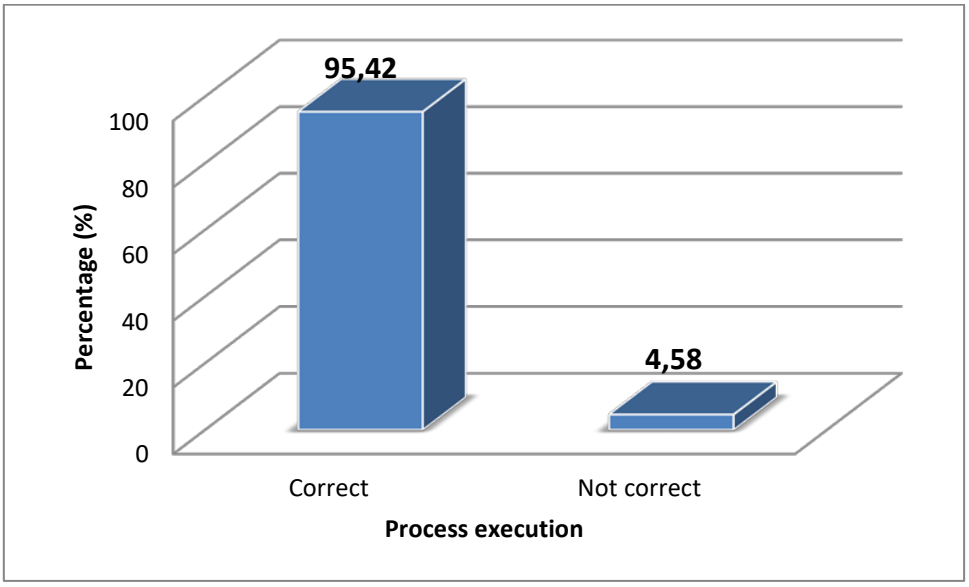


Figure 23: Global results of the experimental validation

Figure 24 shows the wrong process executions divided into three groups depending on the condition which did not get fulfilled. In some cases, more than one condition is broken, so the same execution can be counted various times.

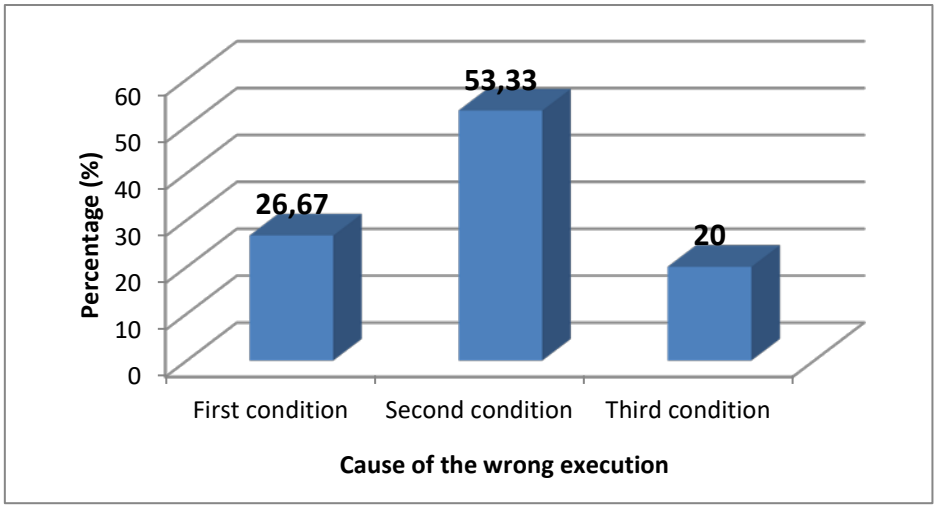


Figure 24: Causes of the wrong executions

As can be seen, more than 50% of wrong executions are due to inefficacies in the proposed study of the process description (branch-by-branch) at business level (algorithm 3). In fact, in synchronization points where only XOR-splits and XOR-joins are involved, synchronization activities are not necessary. As at these points only one output branch and one input branch are considered, no synchronization activity is required (i.e. it is not necessary to wait for other branches being executed in parallel or running various branches at the same time) and the synchronization points only exist virtually (in practice only two flows collide at the same point). Thus, these synchronization points are in fact second order points which introduce an orchestration cost which is not considered in the algorithm. In that way, the synergies of executing an entire branch in a unique location are ignored. If the branch which is going to get active was known previously, then the best execution scheme could be calculated. However, this selection is done dynamically, and the proposed algorithm cannot evaluate these situations. Nevertheless, in absolute terms, these processes represent only a 2.38% of cases (the 53% of the wrong execution -a 4.5% of the total cases-).

Moreover, in 20% of cases, wrong executions are due to problems at the probabilistic execution step. In the total amount of cases, wrong executions appear when data collection services were involved. These problems are generated by experimental errors which influence the system during a time longer than the time employed by the probabilistic execution engine in executing the corresponding task. Thus, despite doing a statistical study (including various realizations and statistical functions), the final results are not correct. Advanced techniques could be necessary in order to address this challenge.

Finally, in 26.7% of cases the actions finally executed by the underlying ad hoc network were not desired by users, or the parameters they indicated were not respected. Causes of these wrong executions are varied. In Figure 25 these causes are showed.

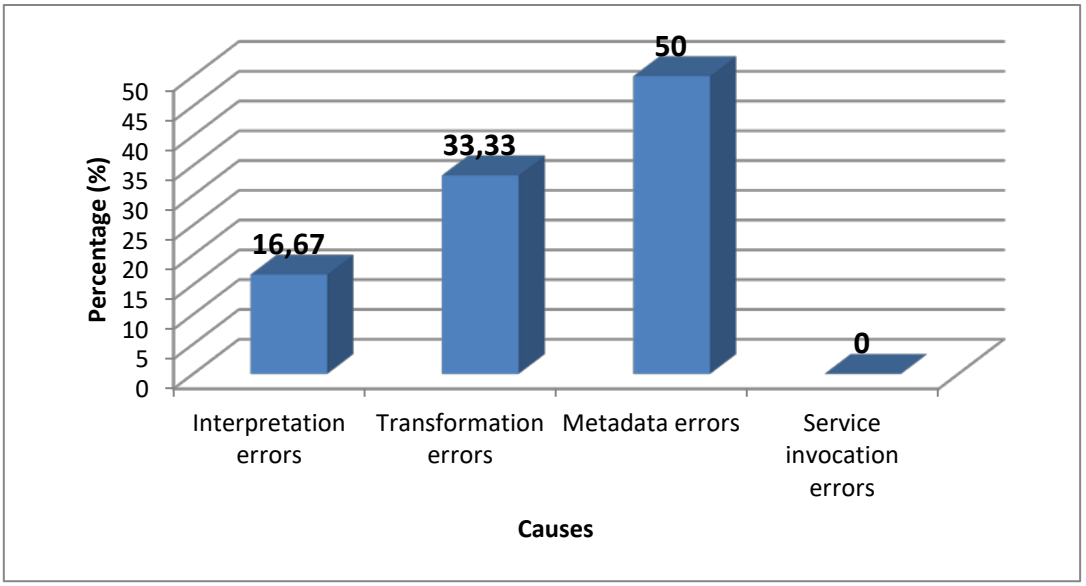


Figure 25: Causes of wrong executions (first condition)

In general, the final cause of the violation of the first condition for a successful execution is the proposed scheme based on multiple successive code and data transformations. Although, in absolute terms, this problem only appears in 1% of cases, if any little error in any of the transformation phases appears (in the metadata or in the process description), the successive transformation stages increase the error, so the final executed actions are not related to the service desired by users. As can be seen in Figure 25 this problem is especially critical in the case of metadata. Parameter transformation (describing the QoS, the statistical execution, etc.) is a complicated and delicate activity, and any minimal a variation in their value represents a great difference at runtime. Recursive feedback transformation algorithms could be necessary in order to avoid these problems.

6. CONCLUSIONS AND FUTURE WORKS

Cyber-Physical Systems have evolved from their origin as advanced embedded devices to a paradigm for the next generation of technological systems. In particular, systems to execute user-defined processes based on service-oriented CPS have emerged as a promising field. Many proposals related to this topic may be found, although none of them truly allow users to define processes in a high-level of abstraction which, later, may be executed in a traditional ad hoc network or pervasive computing infrastructure.

This work proposes a complete scheme (based on CPS paradigm) which covers this gap. The proposal unifies previous works (such as papers related to code transformation and service provision in CPS), and describes new solutions. Contributions include an algorithm for process execution orchestration, execution algorithms based on the concept of guaranteed QoS and probabilistic techniques to wean the random behavior of physical devices from the hard definition of computational processes.

An experimental validation was performed in order to evaluate the proposal. Results show a 95% of user-defined processes (at prosumer level) are correctly executed. Additionally, orchestration algorithms only fail in a 2% of cases, due to the use of complex XOR-splits and XOR-joins. On the other hand, advanced statistical techniques should be implemented in order to avoid the effect of long-duration experimental errors. Moreover, corrective algorithms in the transformation stages should be also considered (especially when transforming metadata). The two last problems only affect the 1% of process executions.

Future works should address the described challenges. Besides, as the proposed experimental validation only considers a first basic implementation of CPS (in a scenario similar to a smart building), future works should also evaluate the performance of the proposal in applications such as traffic flow management (smart cities), electric power generation management and personalized health care devices. Finally, advanced studies about the performance of the proposed framework should also be carried out: scalability of fault resiliency are parameters to be investigated.

REFERENCES

[1] Abid, H., Phuong, L. T. T., Wang, J., Lee, S., & Qaisar, S. (2011, October). V-Cloud: vehicular cyber-physical systems and cloud computing. In Proceedings of the 4th International Symposium on Applied Sciences in Biomedical and Communication Technologies (p. 165). ACM.

[2] Al Ridhawi, Y., & Karmouch, A. (2015). Decentralized plan-free semantic-based service composition in mobile networks. IEEE Transactions on Services Computing, 8(1), 17-31.

[3] Appel, S., Frischbier, S., Freudenreich, T., & Buchmann, A. (2013). Event stream processing units in business processes. In Business Process Management (pp. 187-202). Springer Berlin Heidelberg.

[4] Bakakeu, J., Schäfer, F., Bauer, J., Michl, M., & Franke, J. (2017). Building Cyber-Physical Systems–A Smart Building Use Case. Smart Cities: Foundations, Principles, and Applications, 605-639.

[5] Boekhold, M., Karkowski, I., Corporaal, H., & Cilio, A. (1999, March). A programmable ANSI C transformation engine. In International Conference on Compiler Construction (pp. 292-295). Springer Berlin Heidelberg.

[6] Bonzini, P., & Pozzi, L. (2006, October). Code transformation strategies for extensible embedded processors. In Proceedings of the 2006 international conference on Compilers, architecture and synthesis for embedded systems (pp. 242-252). ACM.

[7] Bordel Sánchez, B., Alcarria, R., Martín, D., & Robles, T. (2015). TF4SM: A Framework for Developing Traceability Solutions in Small Manufacturing Companies. Sensors, 15(11), 29478-29510.

[8] Bordel, B., Alcarria, R., Martín, D., Robles, T., & de Rivera, D. S. (2016). Self-configuration in humanized Cyber-Physical Systems. Journal of Ambient Intelligence and Humanized Computing, 1-12

[9] Bordel, B., Alcarria, R., Robles, T., & Martín, D. (2017). Cyber–physical systems: Extending pervasive sensing from control theory to the Internet of Things. Pervasive and Mobile Computing, 40, 156-184.

[10] Childers, B., Davidson, J. W., & Soffa, M. L. (2003, April). Continuous compilation: A new approach to aggressive and adaptive code transformation. In Parallel and Distributed Processing Symposium, 2003. Proceedings. International (pp. 10-pp). IEEE

[11] Clauß, S., & Schulte, A. (2014, October). Task delegation in an agent supervisory control relationship capability awareness in a cognitive agent. In 2014 IEEE International Conference on Systems, Man, and Cybernetics (SMC) (pp. 825-830). IEEE.

[12] Conti, M., Das, S. K., Bisdikian, C., Kumar, M., Ni, L. M., Passarella, A., ... & Zambonelli, F. (2012). Looking ahead in pervasive computing: Challenges and opportunities in the era of cyber–physical convergence. Pervasive and Mobile Computing, 8(1), 2-21.

[13] Czarnecki, K., & Helsen, S. (2003, October). Classification of model transformation approaches. In Proceedings of the 2nd OOPSLA Workshop on Generative Techniques in the Context of the Model Driven Architecture (Vol. 45, No. 3, pp. 1-17).

[14] Delsing, J., Carlsson, O., Arrigucci, F., Bangemann, T., Hübner, C., Colombo, A. W., ... & Kyusakov, R. (2014). Migration of SCADA/DCS Systems to the SOA Cloud. In Industrial Cloud-Based Cyber-Physical Systems (pp. 111-135). Springer International Publishing.

[15] Derler, P., Lee, E. A., Tripakis, S., & Törngren, M. (2013, April). Cyber-physical system design contracts. In Proceedings of the ACM/IEEE 4th International Conference on Cyber-Physical Systems (pp. 109-118). ACM.

[16] Griffiths, N. (2005, July). Task delegation using experience-based multi-dimensional trust. In Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems (pp. 489-496). ACM.

[17] Hat, R. (2012). libvirt: The virtualization API. Available online: http://libvirt. org.

[18] Kane, M., Zhu, D., Hirose, M., Dong, X., Winter, B., Häckell, M., ... & Swartz, A. (2014, April). Development of an extensible dual-core wireless sensing node for cyber-physical systems. In SPIE Smart Structures and Materials+ Nondestructive Evaluation and Health Monitoring (pp. 90611U-90611U). International Society for Optics and Photonics.

[19] Kim, J., Lakshmanan, K., & Rajkumar, R. R. (2012, April). Rhythmic tasks: A new task model with continually varying periods for cyber-physical systems. In Proceedings of the 2012 IEEE/ACM Third International Conference on Cyber-Physical Systems (pp. 55-64). IEEE Computer Society.

[20] Kivity, A., Kamay, Y., Laor, D., Lublin, U., & Liguori, A. (2007, July). kvm: the Linux virtual machine monitor. In Proceedings of the Linux symposium (Vol. 1, pp. 225-230).

[21] Koubaa, A., Andersson, B.: A vision of cyber-physical internet. In: Proceedings of the Workshop of Real-Time Networks (RTN 2009), Satellite Workshop to (ECRTS 2009)

[22] Kyusakov, R., Eliasson, J., Delsing, J., van Deventer, J., & Gustafsson, J. (2013). Integration of wireless sensor and actuator nodes with IT infrastructure using service-oriented architecture. IEEE Transactions on industrial informatics, 9(1), 43-51.

[23] Lee, E. A. (2006, October). Cyber-physical systems-are computing foundations adequate. In Position Paper for NSF Workshop On Cyber-Physical Systems: Research Motivation, Techniques and Roadmap (Vol. 2).

[24] Lin, K. J., & Panahi, M. (2010, July). A real-time service-oriented framework to support sustainable cyber-physical systems. In 2010 8th IEEE International Conference on Industrial Informatics (pp. 15-21). IEEE.

[25]Liu, L., Zhao, S., Yu, Z., & Dai, H. (2015). A big data inspired chaotic solution for fuzzy feedback linearization model in cyber-physical systems. Ad Hoc Networks, 35, 97-104.

[26]Maier, S., Herrscher, D., & Rothermel, K. (2007). Experiences with node virtualization for scalable network emulation. Computer Communications, 30(5), 943-956.

[27] Marrella, A., & Mecella, M. (2017). Adaptive Process Management in Cyber-Physical Domains. In Advances in Intelligent Process-Aware Information Systems (pp. 15-48). Springer International Publishing.

[28] Moldovan, D., Copil, G., & Dustdar, S. (2017). Elastic systems: Towards cyber-physical ecosystems of people, processes, and things. Computer Standards & Interfaces.

[29] Morales, A., Alcarria, R., Martin, D., & Robles, T. (2014). Enhancing evacuation plans with a situation awareness system based on end-user knowledge provision. Sensors, 14(6), 11153-11178.

[30]Muccini, H., Sharaf, M., & Weyns, D. (2016, May). Self-adaptation for cyber-physical systems: a systematic literature review. In Proceedings of the 11th International Workshop on Software Engineering for Adaptive and Self-Managing Systems (pp. 75-81). ACM.

[31]Mueller, M. (2015). Reducing Hazards in Multiagent Task Delegation. In Wirtschaftsinformatik (pp. 1679-1693).

[32] National institute of Standards and Technology (NIST). (June 2014). DRAFT Framework for Cyber-Physical Systems https://s3.amazonaws.com/nist-sgcps/cpspwg/pwgglobal/CPS_PWG_Draft_Framework_for_Cyber-Physical_Systems_Release_0_8_September_2015.pdf (accessed on 19 August, 2016)

[33] Qiao, L., Kao, S., & Zhang, Y. (2011). Manufacturing process modelling using process specification language. The International Journal of Advanced Manufacturing Technology, 55(5-8), 549-563.

[34] Rajkumar, R. R., Lee, I., Sha, L., & Stankovic, J. (2010, June). Cyber-physical systems: the next computing revolution. In Proceedings of the 47th Design Automation Conference (pp. 731-736). ACM.

[35] Reuillon, R., Chuffart, F., Leclaire, M., Faure, T., Dumoulin, N., & Hill, D. (2010, June). Declarative task delegation in OpenMOLE. In High performance computing and simulation (hpcs), 2010 international conference on (pp. 55-62). IEEE.

[36] Richardson, L., & Ruby, S. (2008). RESTful web services. " O'Reilly Media, Inc.".

[37] Robles, T. S., Alcarria, R., Morales, A., & Martín, D. (2015). Supporting variability dependencies for rule–based service compositions in prosumer environments. International Journal of Web and Grid Services, 11(1), 57-77.

[38] Seiger, R., Huber, S., & Schlegel, T. (2016). Toward an execution system for self-healing workflows in cyber-physical systems. Software & Systems Modeling, 1-22.

[39] Sikora, A. (2005). IEEE802. 15.4 and ZigBe for Sensor and Actuator Networks. In Embedded World Conference (pp. 189-197).

[40] Simpson, B., & Toussi, F. (2004). Hsqldb User Guide.

[41]Sungur, C. T., Spiess, P., Oertel, N., & Kopp, O. (2013, July). Extending bpmn for wireless sensor networks. In 2013 IEEE 15th Conference on Business Informatics (pp. 109-116). IEEE.

[42] Sztipanovits, J., Koutsoukos, X., Karsai, G., Kottenstette, N., Antsaklis, P., Gupta, V., ... & Wang, S. (2012). Toward a science of cyber–physical system integration. Proceedings of the IEEE, 100(1), 29-44.

[43] Tamarit, S., Marino, J., Vigueras, G., & Carro, M. (2016, March). Towards a Semantics-Aware Code Transformation Toolchain for Heterogeneous Systems. In First Workshop on Program Transformation for Programmability in Heterogeneous Architectures (PROHA'16).

[44] Tan, Y., Goddard, S., & Perez, L. C. (2008). A prototype architecture for cyber-physical systems. ACM Sigbed Review, 5(1), 26.

[45] Tranquillini, S., Spieß, P., Daniel, F., Karnouskos, S., Casati, F., Oertel, N., ... & Voigt, T. (2012, September). Process-based design and integration of wireless sensor network applications. In International Conference on Business Process Management (pp. 134-149). Springer Berlin Heidelberg.

[46] Van Der Aalst, W. M., Aldred, L., Dumas, M., & ter Hofstede, A. H. (2004, June). Design and implementation of the YAWL system. In International Conference on Advanced Information Systems Engineering (pp. 142-159). Springer Berlin Heidelberg.

[47] Van Der Aalst, W. M., & Ter Hofstede, A. H. (2005). YAWL: yet another workflow language. Information systems, 30(4), 245-275.

[48] Wan, J., Chen, M., Xia, F., Di, L., & Zhou, K. (2013). From machine-to-machine communications towards cyber-physical systems. Computer Science and Information Systems, 10(3), 1105-1128

[49] Wan, J., Zhang, D., Zhao, S., Yang, L. T., & Lloret, J. (2014). Context-aware vehicular cyber-physical systems with cloud support: architecture, challenges, and solutions. IEEE Communications Magazine, 52(8), 106-113.

[50] Wu, J., & Stojmenovic, I. (2004). Ad hoc networks. COMPUTER-IEEE COMPUTER SOCIETY-, 37(2), 29-31.

[51] Wu, F. J., Kao, Y. F., & Tseng, Y. C. (2011). From wireless sensor networks towards cyber physical systems. Pervasive and Mobile Computing, 7(4), 397-413.

[52] Xia, F., Ma, L., Dong, J., & Sun, Y. (2008, July). Network QoS management in cyber-physical systems. In Embedded Software and Systems Symposia, 2008. ICESS Symposia'08. International Conference on (pp. 302-307). IEEE.

[53] Yue, X., Cai, H., Yan, H., Zou, C., & Zhou, K. (2015). Cloud-assisted industrial cyber-physical systems: an insight. Microprocessors and Microsystems, 39(8), 1262-1270.

[54] Zeng, Y., Li, D., & Vasilakos, A. V. (2013). Real-time data report and task execution in wireless sensor and actuator networks using self-aware mobile actuators. Computer Communications, 36(9), 988-997.

[55] Zhang, J., Lin, Y., & Gray, J. (2005). Generic and domain-specific model refactoring using a model transformation engine. In Model-driven Software Development (pp. 199-217). Springer Berlin Heidelberg.

[56] Zhang, F., Szwaykowska, K., Wolf, W., & Mooney, V. (2008, November). Task scheduling for control oriented requirements for cyber-physical systems. In Real-Time Systems Symposium, 2008 (pp. 47-56). IEEE.

[57] Zhang, W., Wen, Y., & Wu, D. O. (2015). Collaborative task execution in mobile cloud computing under a stochastic wireless channel. IEEE Transactions on Wireless Communications, 14(1), 81-93.

[58] Choi, I., Song, M., Park, C., & Park, N. (2003). An XML-based process definition language for integrated process management. Computers in Industry, 50(1), 85-102.

[59] Duarte, F. J., Machado, R. J., & Fernandes, J. M. (2012, January). BIM: a methodology to transform business processes into software systems. In International Conference on Software Quality (pp. 39-58). Springer, Berlin, Heidelberg.

[60] Santos, N., Duarte, F. J., Machado, R. J., & Fernandes, J. M. (2013, January). A transformation of business process models into software-executable models using mda. In International Conference on Software Quality (pp. 147-167). Springer, Berlin, Heidelberg.

[61] Tao, F., Cheng, Y., Zhang, L., & Nee, A. Y. (2017). Advanced manufacturing systems: socialization characteristics and trends. Journal of Intelligent Manufacturing, 28(5), 1079-1094.

[62] Alcarria, R., Bordel, B., Manso, M. Á., Iturrioz, T., & Pérez, M. (2018, January). Analyzing UAV-Based Remote Sensing and WSN Support for Data Fusion. In International Conference on Information Theoretic Security (pp. 756-766). Springer, Cham.

[63] Perera, C., Zaslavsky, A., Christen, P., & Georgakopoulos, D. (2014). Sensing as a service model for smart cities supported by internet of things. Transactions on Emerging Telecommunications Technologies, 25(1), 81-93.

[64] Sefraoui, O., Aissaoui, M., & Eleuldj, M. (2012). OpenStack: toward an open-source solution for cloud computing. International Journal of Computer Applications, 55(3).

[65] Cirani, S., Picone, M., Gonizzi, P., Veltri, L., & Ferrari, G. (2015). Iot-oas: An oauth-based authorization service architecture for secure services in iot scenarios. IEEE sensors journal, 15(2), 1224-1234.

[66] Kukla, T., Kiss, T., Terstyanszky, G., & Kacsuk, P. (2008, November). A general and scalable solution for heterogeneous workflow invocation and nesting. In Workflows in Support of Large-Scale Science, 2008. WORKS 2008. Third Workshop on (pp. 1-8). IEEE.

[67] Schunselaar, D. M., Verbeek, H. M. W., Reijers, H. A., & van der Aalst, W. M. (2014, September). YAWL in the cloud: supporting process sharing and variability. In International Conference on Business Process Management (pp. 367-379). Springer, Cham.