

# WHY BPEL IS NOT THE HOLY GRAIL OF THE BPM?

## 1 BACK TO THEORY: ABOUT PARALLELISM AND STRUCTUREDNESS IN PROGRAMMING LANGUAGES

Most developers and BPM users may believe that BPEL [1] is a structured language since it is basically based on blocks much as traditional languages such as Java, C among others. This comes from one part of its origin: Microsoft's XLANG which was block based. Anyhow, its other origin part --- namely, IBM's WSFL is of a great importance for the following discussion as it was graph based (hence unstructured). Therefore, we find in BPEL a mix of structuredness (blocks) and unstructuredness (control-links and events). Those last constructions introduce a bit of unstructuredness into a world of structuredness... The conclusion is that BPEL is *not* a structured languages even if it *looks like* a structured language.

On the other side, BPMN is a flow-chart notation which is naturally unstructured. No doubt about this. In chapter 11, page 137 of the BPMN specification [2], a direct mapping from BPMN to BPEL is provided. Therefore, some BPMN editors (and users) may believe that BPMN is a simple GUI for the underlying BPEL language. Nevertheless, in the BPMN FAQ available at <http://www.bpmn.org/Documents/FAQ.htm>, one can read:

*"By design there are some limitations on the process topologies that can be described in BPEL, so it is possible to represent processes in BPMN that cannot be mapped to BPEL".*

This post, will give some fundamental details to that very important statement. But let's focus first on structured versus unstructured languages. Why is it so important? The main point, is that it is much harder to perform code analysis of an unstructured language than of a structured one (such as Java, C, and most --- if not all --- widely used programming languages). Code analysis has a wide range of applications, from error checking (e.g. compilers), to bug detection (e.g. findbugs, deadlock detection, ...) and quality checking (e.g. check style).

An important theorem from Böhm and Jacopini [3] and vulgarized on Wikipedia at [http://en.wikipedia.org/wiki/Structured\\_program\\_theorem](http://en.wikipedia.org/wiki/Structured_program_theorem) states that every computable function can be implemented in a programming language that combines subprograms in only three specific ways. These three control structures are:

- executing one subprogram, and then another subprogram (sequence);
- executing one of two subprograms according to the value of a boolean variable (selection);
- executing a subprogram until a boolean variable is true (iteration).

This basically means that any (unstructured) flow-chart can be transformed into a structured one. This forms the basis of the Dijkstra paper called: "Go To Statement

Considered Harmful" [4].

There is still a debate on whether we should allow unstructured programming language or not. Facts are nevertheless that:

1. most student in the world are taught structured programming;
2. most widely used programming languages are (non-strict) structured programming languages;
3. most unstructured programming languages had introduced some structured constructions (BASIC, COBOL, FORTRAN).

So, in general, most programmers do focus on structured programming but sometimes use unstructured constructions (goto, jump, break, exceptions) for various reasons (mainly readability, maintenance, sometimes performance).

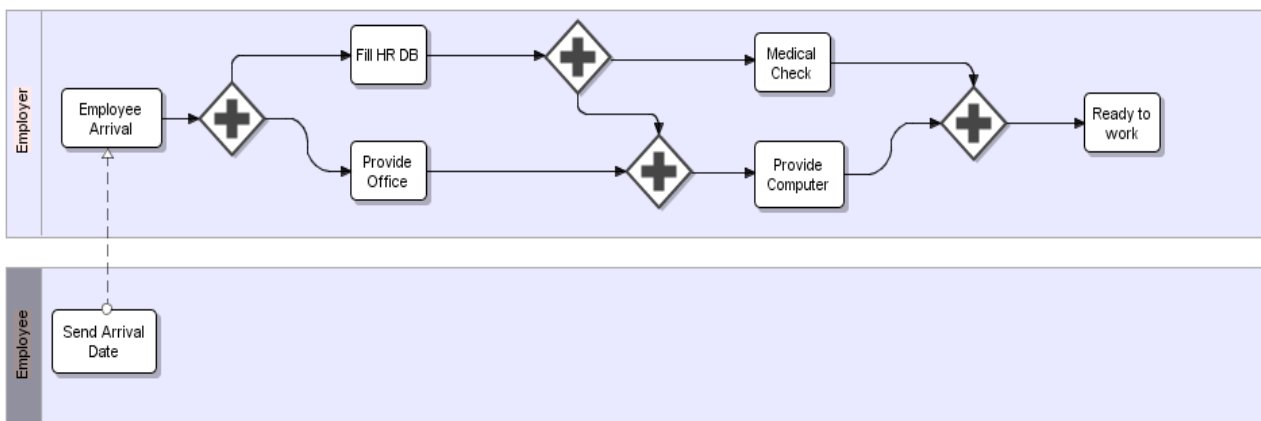
## 1 BUSINESS ANALYSTS WRITE PARALLEL AND UNSTRUCTURED PROCESSES

A business analyst (also called BPM end-user) has to deal with the real world<sup>1</sup> which is in essence, not only unstructured, but highly parallel.

This has two implications:

1. BPM end-users are usually not computer engineers, neither computer scientists: they design business processes using flow-charts notation (this is the most natural) for them, hence, unstructured (and parallel, this is important);
2. in the face of parallelism, unstructuredness is more expressive than structuredness.

Point 2, is of a great importance and has been formally proven by Kiepuszewski *et al.* [5]. Fact is that there are parallel unstructured workflows that cannot be expressed into a parallel structured one. And those cases, are quite simple to find actually. Consider this example using the BPMN notation (made using the Intalio BPMN designer):



Note that a separate pool (called pool0) has to be created in order to validate that diagram in Intalio. This seems to be related to the underlying BPEL format in which BPMN diagrams are exported natively. We will speak about that problem later. The main point here is to

<sup>1</sup> One may say that programmers also have to deal with the real world. But they are definitely not at the same abstraction level (what would be the point otherwise defining specific notations for analysts?).

focus on the pool called 'Pool' and that contains 6 activities. When an employee arrived in a new company, a workflow is started. Basically, the Human Resources database has to be filled. At the same time, an office has to be provided. As soon as the Human Resources has been filled, the employee can move forward to perform a medical check. During that time, a computer has to be provided. This is only possible when both an office has been set up and when an account has been created in the information system from the human resource database. When both the computer and the medical check has been performed, the employee is ready to work. Of course, you may want to model this simple process differently. But the point I want to make here, is that you cannot define a structured parallel workflow that is equivalent<sup>2</sup> to this one, I mean, you will never be able to! We will use this simple example throughout this document.

As a first conclusion from this study we have:

- developers naturally write their programs using sequential structured constructions (blocks);
- BPM users naturally design their processes using unstructured and parallel constructions (graph);
- unstructured and parallel workflows are more expressive than structured parallel ones.

There are workflows that BPM users will design that you definitely cannot express equivalently into a structured parallel workflow. Worse, also in [5], the author shows that even when the transformation from a parallel unstructured workflow to a parallel structured one is possible, it requires the addition of several variables and/or nodes, so that the final result is almost unreadable to the end user. We will speak soon about that readability problem.

## 2 TRANSFORMING BPMN TO BPEL

In the article [8], authors proposed an automatic translation from BPMN to (readable) BPEL. They define a subset of BPMN due to unclear semantic in the specification. Hence, OR gateway, and error intermediate events are just not considered in their paper. This is a problem if the workflow designed contains multiple end events since they are not considered by the transformation tool and the standard way to convert from a multiple end events workflow to a single one is by using OR gateways. The algorithm basically works like this:

1. it tries to find in the graph, well known pattern that maps directly to BPEL structure (sequence, flow, pick, while,...). For each of them, it replaces the component by a simple task containing the mapped BPEL code.
2. Then, it tries to find some quasi-structured component. It transform them in such a way that most of the component is mapped to direct BPEL structure.
3. Then, it searches for acyclic BPMN submodels (containing only sequence flows and parallel gateways). For them, it uses BPEL control-links.
4. Finally, for the rest, BPEL events are used.

---

<sup>2</sup> Of course we have to define what 'equivalent' means. Usually, in the processes world, the formally defined bisimulation [6] is used. But since it cannot make a difference between a parallel execution and its sequential simulation -- something that BPM users want, the notion of fully parallel bisimulation [7] is used instead.

The result, is an "as readable as possible" equivalent BPEL process. Note that using events in BPEL, we can always transform BPMN to BPEL. The problem is that the generated code is not readable at all. The conclusion is that an algorithm for transforming any BPMN diagram into an equivalent<sup>3</sup> BPEL process that is "as readable as possible"<sup>4</sup> does exist.

## THE INTALIO USE CASE

Note that this is certainly not the algorithm used by the Intalio BPM v2.0 solution for their transformation from BPMN to BPEL as shown by the simple example below. Taking the previous parallel unstructured BPMN diagram (page 2), the Intalio solution transforms it into the BPEL process below:

```
<?xml version="1.0" encoding="UTF-8"?>
<bpel:process
  xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
  xmlns:wSDL="http://schemas.xmlsoap.org/wSDL/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:vprop="http://docs.oasis-open.org/wsbpel/2.0/varprop"
  xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:this="http://example.com/Unstructured/Employer"
  xmlns:Employee="http://example.com/Unstructured/Employee"
  xmlns:diag="http://example.com/Unstructured"
  xmlns:xml="http://www.w3.org/XML/1998/namespace"
  xmlns:bpmn="http://www.intalio.com/bpms"
  xmlns:atomic="http://ode.apache.org/atomicScope"
  queryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath2.0"
  expressionLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath2.0"
  bpmn:label="Employer" bpmn:id="_TGthsJkdEd29aqy0ek4Sxw" name="Employer"
  targetNamespace="http://example.com/Unstructured/Employer">
  <bpel:import namespace="http://example.com/Unstructured"
    location="Unstructured.wSDL"
    importType="http://schemas.xmlsoap.org/wSDL/" />
  <bpel:import namespace="http://example.com/Unstructured/Employer"
    location="Unstructured-Employer.wSDL"
    importType="http://schemas.xmlsoap.org/wSDL/" />
  <bpel:partnerLinks>
    <bpel:partnerLink name="employeeAndEmployerPlkVar"
      partnerLinkType="diag:EmployeeAndEmployer"
      myRole="Employer_for_Employee" />
  </bpel:partnerLinks>
  <bpel:variables>
    <bpel:variable name="thisEmployee_ArrivalRequestMsg"
      messageType="this:Employee_ArrivalRequest" />
  </bpel:variables>
  <bpel:sequence>
    <bpel:receive partnerLink="employeeAndEmployerPlkVar"
      portType="this:ForEmployee" operation="Employee_Arrival"
      variable="thisEmployee_ArrivalRequestMsg" createInstance="yes"
      bpmn:label="Employee Arrival" bpmn:id="_THp84JkdEd29aqy0ek4Sxw">
    </bpel:receive>
    <bpel:flow bpmn:label="GatewayParallel"
      bpmn:id="_DHLtcJkeEd29aqy0ek4Sxw">
    <bpel:sequence>
      <bpel:empty bpmn:label="Fill HR DB"

```

3 According to the notion of fully parallel bisimulation.

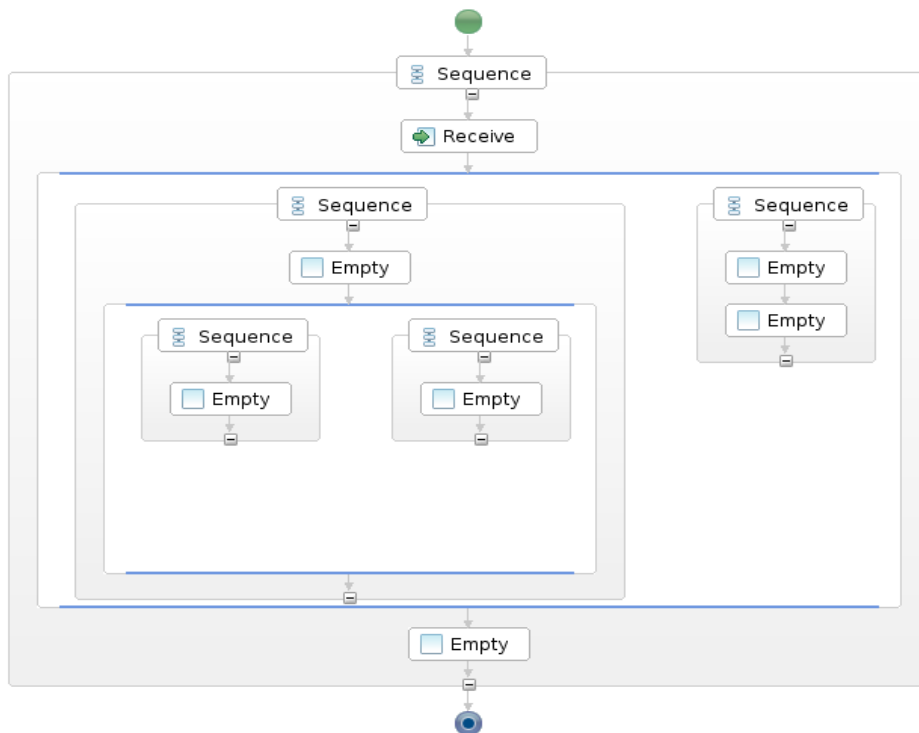
4 According to the common idea of a human readable BPEL, *i.e.* made of block wherever possible.

```

    bpmn:id="_hCvXsJkdEd29aqy0ek4Sxw" />
  <bpel:flow bpmn:label="GatewayParallel"
    bpmn:id="_H2UWEJkeEd29aqy0ek4Sxw">
    <bpel:sequence>
      <bpel:empty bpmn:label="Medical Check"
        bpmn:id="_ivks8JkdEd29aqy0ek4Sxw" />
    </bpel:sequence>
    <bpel:sequence>
      <bpel:empty bpmn:label="Provide Computer"
        bpmn:id="_lXApQJkdEd29aqy0ek4Sxw" />
    </bpel:sequence>
  </bpel:flow>
</bpel:sequence>
<bpel:sequence>
  <bpel:empty bpmn:label="Provide Office"
    bpmn:id="_iHEigJkdEd29aqy0ek4Sxw" />
  <bpel:empty bpmn:label="Provide Computer"
    bpmn:id="_lXApQJkdEd29aqy0ek4Sxw" />
</bpel:sequence>
</bpel:flow>
<bpel:empty bpmn:label="Ready to work"
  bpmn:id="_nh6akJkdEd29aqy0ek4Sxw" />
</bpel:sequence>
</bpel:process>

```

To get a better picture of the transformation output, we opened the process into the Eclipse BPEL designer:



First, for some reasons, activities labeled 'Fill HR Db', 'Medical Check', and so on, are not displayed. Anyway, we can see from BPEL source code that BPMN activity 'Employee Arrival' has been transformed into a 'Receive' BPEL operation. Therefore it is rather strange to count 7 activities ('Receive' and 6 other 'Empty' activities) while our original process contained only 6 of them. Looking to the BPEL source code, we can see that activity 'Provide Computer' has been duplicated. In some ways, this is good for the employee: it will

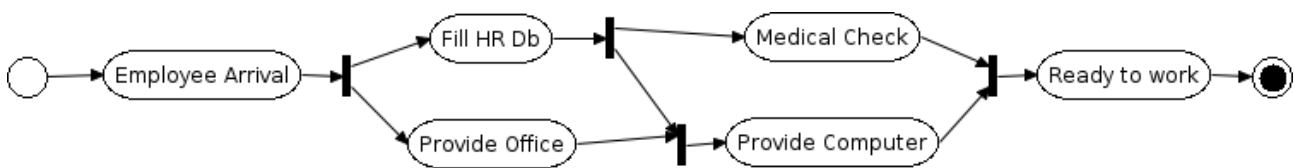
get two computers in his office!

Whether the Intalio BPMN2BPEL transformation algorithm produces a readable BPEL or not, is not the problem here: it is simply wrong. You can hardly imagine the result on diagrams produced by professional BPM designers focusing on real business processes since they are highly parallel and unstructured in essence.

### 3 THE READABILITY PROBLEM

The claim in [9] is that BPEL is not readable by the end-user. So there is a need for a high-level language in which processes are designed. But some informations will have to be entered for it to be actually executed. This will have to be entered in some ways into the resulting BPEL code. So it is important to generate a readable BPEL code. In the case where all informations are entered directly into the editor/designer, at least for debugging purposes, the BPEL code should be as readable as possible where readable means: using BPEL direct straightforward structure (sequence, flow, pick, wait, etc...) as much as possible.

On this readability issue, let's introduce the JWT eclipse sub-project<sup>5</sup>. Its aim is to provide a toolkit for Workflow management (designer, transformations, simulations and connections to engine). JWT currently uses the UML Activity Diagram<sup>6</sup> notation (UML-AD for short) for designing workflows. UML-AD is strictly equivalent (in terms of expressive power) to BPMN (see [10] for details). So, using an UML-AD notation, we can represent the previous BPMN diagram in JWT:



JWT has been developed to be extensible, and provides different transformation plugins. One of them is a UML-AD2BPEL transformation provided as a research project from the University of Augsburg. The BPEL transformation plugin outputs a BPEL-WS-1.1 document that is 518 lines long (provided at the end of this document). Note that neither Eclipse BPEL Designer nor Intalio Designer were able to display properly this BPEL file. We used Netbeans for that purpose. The diagram produced is too complex to be presented here (it is given partially at the end of this document however). The JWT2BPEL transformation tool uses BPEL events extensively in order to get the equivalent<sup>7</sup> BPEL representation. For the courageous readers who tried to read and understand the resulting BPEL file, it should be obvious that it is very hard.

So the simple parallel and unstructured process expressed using pure workflow notation such as BPMN or UML-AD is hardly representable into a “readable” BPEL format. And this is a general fact, not a specific-to-this simple-process one.

Worse is the BPMN-to-BPEL round-trip engineering problem: (from Wikipedia) “generating

<sup>5</sup> <http://www.eclipse.org/jwt/>

<sup>6</sup> See [http://en.wikipedia.org/wiki/Activity\\_diagram](http://en.wikipedia.org/wiki/Activity_diagram) for a very short overview.

<sup>7</sup> By the way, proving formally that the resulting BPEL file is really equivalent (bisimulation) to the original BPMN diagram is left as an exercise... ;-)

BPEL code from BPMN diagrams and maintaining the original BPMN model and the generated BPEL code synchronized, in the sense that any modification to one is propagated to the other”. Definitely, using BPEL as the final format for executing business processes expressed using a natural workflow notation such as BPMN leads to troubles.

Note that making a BPMN diagram out of a BPEL process seems to be much easier than the reverse: transforming structured elements to unstructured elements is straightforward. Note that such a transformation --- BPEL2BPMN --- is provided in Intalio in the Java class available [here](#).

It seems to be in the core of STP and it is (currently) not fully BPEL compliant seeing the comments in the class

```
/*  
 * Very basic sample that generates BPMN out of a BPEL file.  
 * The BPEL parsed is a small subset of the bpeL spec:  
 * scope, assign, receive, reply, invoke, flow, sequence.  
 * ...  
 */
```

Still, importing the BPEL file generated by the Intalio Designer itself does not work for obscure reasons. Nevertheless, the round-trip problem is about synchronizing two quite different representations of the same process: one in BPMN and one in BPEL.

## 4 CONCLUSION

First, we have clarified a common misunderstanding: BPEL is not a structured language, but it is based on a structured language (block-based). In some ways, BPEL is much closer to a standard language such as Java than to a natural workflow notation such as BPMN (which is graph-based). Up to now, programmers deal directly with their language. Integrated Development Environments are used to simplify several recurrent steps such as compiling, refactoring, testing and so on. But, programmers “speak” their language directly. We claim that it should be the same with BPEL. IDE can only simplify the programming (note that we don't use the term “designing” here). But BPEL programmers will have to “speak” BPEL in order to use it and to make something useful out of it. The question of whether BPEL is “speakable” or not by general technicians --- as Java can be --- is out of scope of this article, but is definitely an interesting question.

For the business analyst however, it is clear that BPEL is not user-friendly. BPEL is hard to read, hard to learn, hard to implement, and most of all, as this is the major end-user concern: hard to hide. We have already noticed that when creating the “Employer” pool in the simple example used in this document, we were constraint to create another pool that should be marked “non-executable” in order to get the BPEL file. Many other BPEL related stuff are present in the Intalio designer that are irrelevant from the point of view of a BPMN analyst: *e.g.* namespaces, web-service invocations, XML data type, among others.

Therefore, we consider BPMN notation as the only currently viable solution for Business Analysts<sup>8</sup>. Nevertheless, many execution details, absent from the BPMN specification --- and

---

<sup>8</sup> UML-AD notation may also be used as its expressive power is equivalent to BPMN. Nevertheless, we think

unknown at design time from the analyst --- will have to be specified before the process can really get executed. These informations are usually site specific (*e.g.* mail server address, task repository), and sometimes implementation dependent (*e.g.* web service, J2EE service or .NET service). These informations are technical, and will be specified by a technician. It is therefore of a great importance that the process skeleton on which the technician will enter those informations is both equivalent to the original BPMN process in terms of execution semantic (bisimulation) and easy to read to ensure that modifications made to the process by the technician during the specification of execution informations do not change the process behavior.

Transformation from BPMN to (readable) BPEL is quite hard to implement, and produces --- when correct --- hardly readable code. By the way, the round-trip problem is even harder. This last problem unless resolved, makes BPEL a very difficult target for the output of a process designed by a real business analyst.

Therefore, we may wonder why BPMN is transformed to BPEL since there exist a graph-based standard that maps directly BPMN constructs --- namely XPD v2.0. With this mapping, XPD v2.0 becomes the natural BPMN persistence file format. Moreover, it specifies behaviors that were only available previously in BPEL such as Web Service invocations and compensations. Of course, one may claim that XPD 2.0 lacks some execution specifications that makes him unsuitable for direct execution. We believe that using the BPEL semantic wherever XPD is under-specified makes room for an engine that can be fully BPMN v2.0, XPD v2.0 and BPEL compliant. This is how Bonita and Orchestra team will implement their next generation of BPM engine. But this is another story, that requires an article of its own... Stay tuned!

I would like to thank the Bonita & Orchestra team for their help and support during the writing of this article, and especially Miguel Valdes-Faura for its review and suggestions.

Pierre Vign ras  
Bull, Architect of an Open World TM  
\*BPM Team\*, Bull R&D  
1, rue de Provence  
38130 Echirolles (France)  
Direct Line: +33-4-76-29-74-06

\*Orchestra\*, The BPEL open source project: <http://orchestra.ow2.org>  
\*Bonita\*, The XPD open source project: <http://bonita.ow2.org>

---

that BPMN is closer to BPM analyst needs.

# APENDIX

## A) JWT2BPEL

The JWT2BPEL transformation outputs the following BPEL file:

```
<?xml version="1.0" encoding="UTF-8"?>
<process name="AtmFrontEnd"
  xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
  xmlns:acc="urn:samples:account"
  xmlns:agi="urn:samples:agilpro"
  xmlns:atm="urn:samples:atm"
  xmlns:bpel="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
  xmlns:tic="urn:samples:ticket"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  targetNamespace="urn:samples:atm">

  <partnerLinks>
    <!-- relationship with the ATM -->
    <partnerLink myRole="FrontEnd" name="atm" partnerLinkType="atm:Atm-Front"/>
    <!-- relationship with the ticket issuer -->
    <partnerLink name="ticket" partnerLinkType="atm:Front-Ticket"
partnerRole="TicketIssuer"/>
    <!-- relationship with the account system -->
    <partnerLink name="account" partnerLinkType="atm:Front-Account"
partnerRole="AccountSystem"/>
    <!-- relationship with the agilpro framework -->
    <partnerLink name="agilpro" partnerLinkType="atm:Front-Agilpro"
partnerRole="AgilproIssuer"/>
    <!-- local control-link relationship between between activities -->
    <partnerLink myRole="localService" name="local" partnerLinkType="localLT"/>
  </partnerLinks>

  <variables>
    <!-- ATM connection request -->
    <variable messageType="atm:connectRequest" name="connectReq"/>
    <!-- ticket creation request -->
    <variable messageType="tic:ticketRequest" name="ticketReq"/>
    <!-- ticket number wrapper -->
    <variable messageType="tic:ticketMessage" name="ticketMsg"/>
    <!-- ATM connection flag -->
    <variable name="connected" type="xsd:boolean"/>
    <!-- customer session flag -->
    <variable name="logged" type="xsd:boolean"/>
    <variable messageType="agi:connectToAgilproRequest"
name="connectToAgilproReq"/>
    <variable messageType="agi:connectToAgilproResponse"
name="connectToAgilproRes"/>
    <variable messageType="agi:setApplicationRequest" name="setApplicationReq"/>
    <variable messageType="agi:setValueToObjectRequest"
name="setValueToObjectReq"/>
    <variable messageType="agi:getValueFromObjectRequest"
name="getValueFromObjectReq"/>
    <variable messageType="agi:getValueFromObjectResponse"
name="getValueFromObjectRes"/>
    <variable messageType="agi:startActionRequest" name="startActionReq"/>
  </variables>
</process>
```

```

    <variable messageType="agi:endActionRequest" name="endActionReq"/>
    <variable messageType="agi:nextActionRequest" name="nextActionReq"/>
    <variable messageType="agi:nextActionResponse" name="nextActionRes"/>
    <variable messageType="agi:checkGuardRequest" name="checkGuardReq"/>
    <variable messageType="agi:checkGuardResponse" name="checkGuardRes"/>
    <variable messageType="agi:disconnectAgilproRequest"
name="disconnectAgilproReq"/>
    <variable messageType="agi:disconnectAgilproResponse"
name="disconnectAgilproRes"/>
    <variable messageType="agi:cancelProcessRequest" name="cancelProcessReq"/>
    <variable messageType="agi:cancelProcessResponse" name="cancelProcessRes"/>
</variables>

<correlationSets>
  <!-- conversation with a connected ATM -->
  <correlationSet name="atmInteraction" properties="atm:ticketId"/>
</correlationSets>

<!-- structure of process -->
<sequence name="mainSequence">
  <!-- receive a connection request -->
  <receive createInstance="yes" name="connectIn" operation="connect"
partnerLink="atm" portType="atm:FrontEnd" variable="connectReq"/>
  <!-- generate a ticket number -->
  <invoke inputVariable="ticketReq" name="createTicket"
operation="createTicket" outputVariable="ticketMsg" partnerLink="ticket"
portType="tic:TicketIssuer">
    <correlations>
      <correlation initiate="yes" pattern="in" set="atmInteraction"/>
    </correlations>
  </invoke>
  <!-- initialize the status flags -->
  <assign name="initConnection">
    <copy>
      <from expression="true()"/>
      <to variable="connected"/>
    </copy>
    <copy>
      <from expression="false()"/>
      <to variable="logged"/>
    </copy>
  </assign>
  <!-- send the ticket number back to the ATM -->
  <reply name="connectOut" operation="connect" partnerLink="atm"
portType="atm:FrontEnd" variable="ticketMsg">
    <correlations>
      <correlation set="atmInteraction"/>
    </correlations>
  </reply>
  <!-- handle the ATM connection -->
  <scope name="connectionUnit" variableAccessSerializable="no">
    <variables>
      <!-- customer log on request -->
      <variable messageType="atm:logOnRequest" name="logOnReq"/>
      <!-- connection status response -->
      <variable messageType="atm:statusResponse" name="statusRsp"/>
    </variables>
    <sequence name="START">
      <receive operation="connectToAgilpro" partnerLink="atm"
portType="atm:FrontEnd" variable="connectToAgilproReq">

```

```

    <correlations>
      <correlation set="atmInteraction"/>
    </correlations>
  </receive>
  <assign name="copy_Ticketnumber" >
    <copy>
      <from expression="string('Successfully connected to Agilpro
Framework!')"/>
      <to part="connectToAgilproOUT" variable="connectToAgilproRes"/>
    </copy>
    <copy>
      <from part="Ticketnumber" variable="connectToAgilproReq"/>
      <to part="Ticketnumber" variable="nextActionReq"/>
    </copy>
  </assign>
  <reply operation="connectToAgilpro" partnerLink="atm"
portType="atm:FrontEnd" variable="connectToAgilproRes"/>
  <sequence name="S6">
    <scope name="EmployeeArrival">
      <sequence>
        <empty/>
        <assign name="set_App_">
          <copy>
            <from part="Ticketnumber" variable="nextActionReq"/>
            <to part="Ticketnumber" variable="setApplicationReq"/>
          </copy>
          <copy>
            <from expression="string('Employee Arrival')"/>
            <to part="ActionNameIN" variable="setApplicationReq"/>
          </copy>

          <copy>
            <from expression="string('')"/>
            <to part="ActivityNameIN" variable="setApplicationReq"/>
          </copy>
        </assign>
        <invoke name="setApplication_" inputVariable="setApplicationReq"
operation="setApplication" partnerLink="agilpro" portType="agi:AgilproIssuer">
          <correlations>
            <correlation pattern="out" set="atmInteraction"/>
          </correlations>
        </invoke>
        <assign name="startAction_EmployeeArrival" >
          <copy>
            <from part="Ticketnumber" variable="nextActionReq"/>
            <to part="Ticketnumber" variable="startActionReq"/>
          </copy>
        </assign>
        <invoke name="startAction_EmployeeArrival"
inputVariable="startActionReq" operation="startAction" partnerLink="agilpro"
portType="agi:AgilproIssuer">
          <correlations>
            <correlation pattern="out" set="atmInteraction"/>
          </correlations>
        </invoke>
        <receive variable="nextActionReq" operation="nextAction"
partnerLink="atm" portType="atm:FrontEnd">
          <correlations>
            <correlation set="atmInteraction"/>
          </correlations>

```

```

    </receive>
    <assign name="reply_EmployeeArrival">
      <copy>
        <from expression="string('EmployeeArrival')"/>
        <to part="nextActionOUT" variable="nextActionRes"/>
      </copy>
    </assign>
    <reply operation="nextAction" partnerLink="atm"
portType="atm:FrontEnd" variable="nextActionRes"/>
    <assign name="endAction_EmployeeArrival">
      <copy>
        <from part="Ticketnumber" variable="nextActionReq"/>
        <to part="Ticketnumber" variable="endActionReq"/>
      </copy>
    </assign>
    <invoke name="endAction_EmployeeArrival"
inputVariable="endActionReq" operation="endAction" partnerLink="agilpro"
portType="agi:AgilproIssuer">
      <correlations>
        <correlation pattern="out" set="atmInteraction"/>
      </correlations>
    </invoke>
  </sequence>
</scope>
<scope name="C5">
  <eventHandlers>
    <onEvent partnerLink="local" portType="localPT" operation
="a(entryC5,F0)">
      <flow name ="C5">
        <invoke partnerLink="local" portType="localPT"
operation="a(F0,ProvideOffice)" /><invoke partnerLink="local" portType="localPT"
operation="a(F0,FillHRDb)" />
      </flow>
    </onEvent>
    <onEvent partnerLink="local" portType="localPT" operation
="a(F0,ProvideOffice)">
      <sequence name ="sProvideOffice">
        <scope name="ProvideOffice">
          <sequence>
            <empty/>
            <assign name="set_App_">
              <copy>
                <from part="Ticketnumber" variable="nextActionReq"/>
                <to part="Ticketnumber" variable="setApplicationReq"/>
              </copy>
              <copy>
                <from expression="string('Provide Office')"/>
                <to part="ActionNameIN" variable="setApplicationReq"/>
              </copy>

              <copy>
                <from expression="string('')"/>
                <to part="ActivityNameIN"
variable="setApplicationReq"/>
              </copy>
            </assign>
            <invoke name="setApplication_"
inputVariable="setApplicationReq" operation="setApplication"
partnerLink="agilpro" portType="agi:AgilproIssuer">
              <correlations>

```

```

        <correlation pattern="out" set="atmInteraction"/>
    </correlations>
</invoke>
<assign name="startAction_ProvideOffice" >
    <copy>
        <from part="Ticketnumber" variable="nextActionReq"/>
        <to part="Ticketnumber" variable="startActionReq"/>
    </copy>
</assign>
<invoke name="startAction_ProvideOffice"
inputVariable="startActionReq" operation="startAction" partnerLink="agilpro"
portType="agi:AgilproIssuer">
    <correlations>
        <correlation pattern="out" set="atmInteraction"/>
    </correlations>
</invoke>
<receive variable="nextActionReq" operation="nextAction"
partnerLink="atm" portType="atm:FrontEnd">
    <correlations>
        <correlation set="atmInteraction"/>
    </correlations>
</receive>
<assign name="reply_ProvideOffice">
    <copy>
        <from expression="string('ProvideOffice')"/>
        <to part="nextActionOUT" variable="nextActionRes"/>
    </copy>
</assign>
<reply operation="nextAction" partnerLink="atm"
portType="atm:FrontEnd" variable="nextActionRes"/>
<assign name="endAction_ProvideOffice">
    <copy>
        <from part="Ticketnumber" variable="nextActionReq"/>
        <to part="Ticketnumber" variable="endActionReq"/>
    </copy>
</assign>
<invoke name="endAction_ProvideOffice"
inputVariable="endActionReq" operation="endAction" partnerLink="agilpro"
portType="agi:AgilproIssuer">
    <correlations>
        <correlation pattern="out" set="atmInteraction"/>
    </correlations>
</invoke>
</sequence>
</scope><invoke partnerLink="local" portType="localPT"
operation="a(ProvideOffice,J0)" />
</sequence>
</onEvent>
<onEvent partnerLink="local" portType="localPT" operation
="a(F1,J0)">
    <sequence name ="sJ0">
        <receive name="a(ProvideOffice,J0)" />
        <invoke partnerLink="local" portType="localPT"
operation="a(J0,ProvideComputer)" />
    </sequence>
</onEvent>
<onEvent partnerLink="local" portType="localPT" operation
="a(J0,ProvideComputer)">
    <sequence name ="sProvideComputer">
        <scope name="ProvideComputer">

```

```

<sequence>
  <empty/>
  <assign name="set_App_">
    <copy>
      <from part="Ticketnumber" variable="nextActionReq"/>
      <to part="Ticketnumber" variable="setApplicationReq"/>
    </copy>
    <copy>
      <from expression="string('Provide Computer')"/>
      <to part="ActionNameIN" variable="setApplicationReq"/>
    </copy>

    <copy>
      <from expression="string('')"/>
      <to part="ActivityNameIN"
variable="setApplicationReq"/>
    </copy>
  </assign>
  <invoke name="setApplication_"
inputVariable="setApplicationReq" operation="setApplication"
partnerLink="agilpro" portType="agi:AgilproIssuer">
    <correlations>
      <correlation pattern="out" set="atmInteraction"/>
    </correlations>
  </invoke>
  <assign name="startAction_ProvideComputer" >
    <copy>
      <from part="Ticketnumber" variable="nextActionReq"/>
      <to part="Ticketnumber" variable="startActionReq"/>
    </copy>
  </assign>
  <invoke name="startAction_ProvideComputer"
inputVariable="startActionReq" operation="startAction" partnerLink="agilpro"
portType="agi:AgilproIssuer">
    <correlations>
      <correlation pattern="out" set="atmInteraction"/>
    </correlations>
  </invoke>
  <receive variable="nextActionReq" operation="nextAction"
partnerLink="atm" portType="atm:FrontEnd">
    <correlations>
      <correlation set="atmInteraction"/>
    </correlations>
  </receive>
  <assign name="reply_ProvideComputer">
    <copy>
      <from expression="string('ProvideComputer')"/>
      <to part="nextActionOUT" variable="nextActionRes"/>
    </copy>
  </assign>
  <reply operation="nextAction" partnerLink="atm"
portType="atm:FrontEnd" variable="nextActionRes"/>
  <assign name="endAction_ProvideComputer">
    <copy>
      <from part="Ticketnumber" variable="nextActionReq"/>
      <to part="Ticketnumber" variable="endActionReq"/>
    </copy>
  </assign>
  <invoke name="endAction_ProvideComputer"
inputVariable="endActionReq" operation="endAction" partnerLink="agilpro"

```

```

portType="agi:AgilproIssuer">
    <correlations>
        <correlation pattern="out" set="atmInteraction"/>
    </correlations>
</invoke>
</sequence>
</scope><invoke partnerLink="local" portType="localPT"
operation="a(ProvideComputer,J1)" />
</sequence>
</onEvent>
<onEvent partnerLink="local" portType="localPT" operation
="a(ProvideComputer,J1)">
    <sequence name ="sJ1">
        <receive name="a(MedicalCheck,J1)" />
        <invoke partnerLink="local" portType="localPT"
operation="a(J1,exitC5)" />
    </sequence>
</onEvent>
<onEvent partnerLink="local" portType="localPT" operation
="a(F0,FillHRDb)">
    <sequence name ="sFillHRDb">
        <scope name="FillHRDb">
            <sequence>
                <empty/>
                <assign name="set_App_">
                    <copy>
                        <from part="Ticketnumber" variable="nextActionReq"/>
                        <to part="Ticketnumber" variable="setApplicationReq"/>
                    </copy>
                    <copy>
                        <from expression="string('Fill HR Db')"/>
                        <to part="ActionNameIN" variable="setApplicationReq"/>
                    </copy>

                    <copy>
                        <from expression="string('')"/>
                        <to part="ActivityNameIN"
variable="setApplicationReq"/>
                    </copy>
                </assign>
                <invoke name="setApplication_"
inputVariable="setApplicationReq" operation="setApplication"
partnerLink="agilpro" portType="agi:AgilproIssuer">
                    <correlations>
                        <correlation pattern="out" set="atmInteraction"/>
                    </correlations>
                </invoke>
                <assign name="startAction_FillHRDb" >
                    <copy>
                        <from part="Ticketnumber" variable="nextActionReq"/>
                        <to part="Ticketnumber" variable="startActionReq"/>
                    </copy>
                </assign>
                <invoke name="startAction_FillHRDb"
inputVariable="startActionReq" operation="startAction" partnerLink="agilpro"
portType="agi:AgilproIssuer">
                    <correlations>
                        <correlation pattern="out" set="atmInteraction"/>
                    </correlations>
                </invoke>

```

```

        <receive variable="nextActionReq" operation="nextAction"
partnerLink="atm" portType="atm:FrontEnd">
        <correlations>
            <correlation set="atmInteraction"/>
        </correlations>
    </receive>
    <assign name="reply_FillHRDb">
        <copy>
            <from expression="string('FillHRDb')"/>
            <to part="nextActionOUT" variable="nextActionRes"/>
        </copy>
    </assign>
    <reply operation="nextAction" partnerLink="atm"
portType="atm:FrontEnd" variable="nextActionRes"/>
    <assign name="endAction_FillHRDb">
        <copy>
            <from part="Ticketnumber" variable="nextActionReq"/>
            <to part="Ticketnumber" variable="endActionReq"/>
        </copy>
    </assign>
    <invoke name="endAction_FillHRDb"
inputVariable="endActionReq" operation="endAction" partnerLink="agilpro"
portType="agi:AgilproIssuer">
        <correlations>
            <correlation pattern="out" set="atmInteraction"/>
        </correlations>
    </invoke>
</sequence>
</scope><invoke partnerLink="local" portType="localPT"
operation="a(FillHRDb,F1)" />
</sequence>
</onEvent>
<onEvent partnerLink="local" portType="localPT" operation
="a(FillHRDb,F1)">
    <flow name ="C5">
        <invoke partnerLink="local" portType="localPT"
operation="a(F1,MedicalCheck)" /><invoke partnerLink="local" portType="localPT"
operation="a(F1,J0)" />
    </flow>
</onEvent>
<onEvent partnerLink="local" portType="localPT" operation
="a(F1,MedicalCheck)">
    <sequence name ="sMedicalCheck">
        <scope name="MedicalCheck">
            <sequence>
                <empty/>
                <assign name="set_App_">
                    <copy>
                        <from part="Ticketnumber" variable="nextActionReq"/>
                        <to part="Ticketnumber" variable="setApplicationReq"/>
                    </copy>
                    <copy>
                        <from expression="string('Medical Check')"/>
                        <to part="ActionNameIN" variable="setApplicationReq"/>
                    </copy>

                    <copy>
                        <from expression="string('')"/>
                        <to part="ActivityNameIN"
variable="setApplicationReq"/>

```

```

        </copy>
    </assign>
    <invoke name="setApplication_"
inputVariable="setApplicationReq" operation="setApplication"
partnerLink="agilpro" portType="agi:AgilproIssuer">
        <correlations>
            <correlation pattern="out" set="atmInteraction"/>
        </correlations>
    </invoke>
    <assign name="startAction_MedicalCheck" >
        <copy>
            <from part="Ticketnumber" variable="nextActionReq"/>
            <to part="Ticketnumber" variable="startActionReq"/>
        </copy>
    </assign>
    <invoke name="startAction_MedicalCheck"
inputVariable="startActionReq" operation="startAction" partnerLink="agilpro"
portType="agi:AgilproIssuer">
        <correlations>
            <correlation pattern="out" set="atmInteraction"/>
        </correlations>
    </invoke>
    <receive variable="nextActionReq" operation="nextAction"
partnerLink="atm" portType="atm:FrontEnd">
        <correlations>
            <correlation set="atmInteraction"/>
        </correlations>
    </receive>
    <assign name="reply_MedicalCheck">
        <copy>
            <from expression="string('MedicalCheck')"/>
            <to part="nextActionOUT" variable="nextActionRes"/>
        </copy>
    </assign>
    <reply operation="nextAction" partnerLink="atm"
portType="atm:FrontEnd" variable="nextActionRes"/>
    <assign name="endAction_MedicalCheck">
        <copy>
            <from part="Ticketnumber" variable="nextActionReq"/>
            <to part="Ticketnumber" variable="endActionReq"/>
        </copy>
    </assign>
    <invoke name="endAction_MedicalCheck"
inputVariable="endActionReq" operation="endAction" partnerLink="agilpro"
portType="agi:AgilproIssuer">
        <correlations>
            <correlation pattern="out" set="atmInteraction"/>
        </correlations>
    </invoke>
</sequence>
</scope><invoke partnerLink="local" portType="localPT"
operation="a(MedicalCheck,J1)" />
    </sequence>
</onEvent>
</eventHandlers>
    <invoke partnerLink="local" portType="localPT"
operation="a(entryC5,F0)" />
</scope>
<scope name="Readytowork">
    <sequence>

```

```

<empty/>
<assign name="set_App_">
  <copy>
    <from part="Ticketnumber" variable="nextActionReq"/>
    <to part="Ticketnumber" variable="setApplicationReq"/>
  </copy>
  <copy>
    <from expression="string('Ready to work')"/>
    <to part="ActionNameIN" variable="setApplicationReq"/>
  </copy>

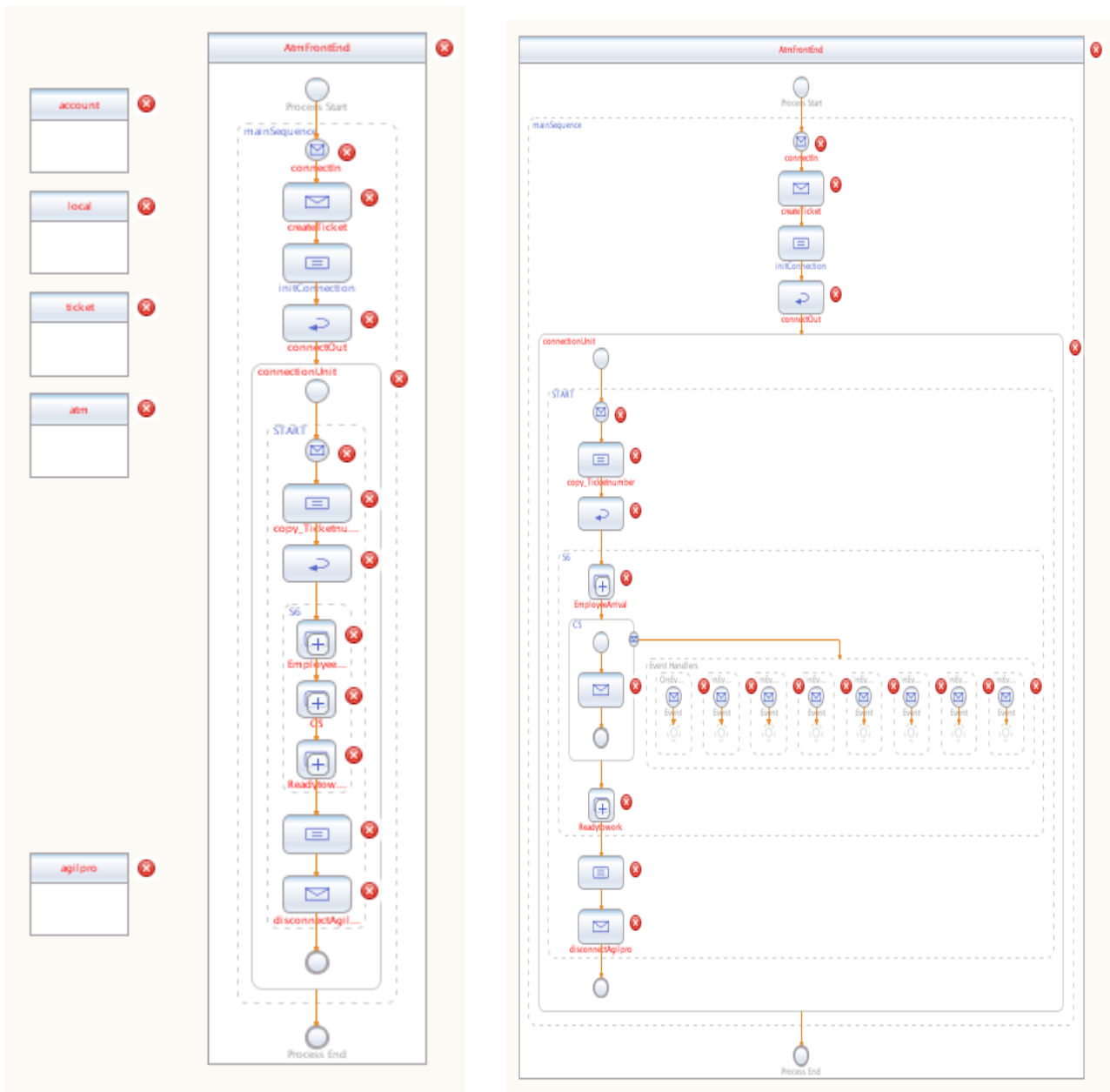
  <copy>
    <from expression="string('')"/>
    <to part="ActivityNameIN" variable="setApplicationReq"/>
  </copy>
</assign>
<invoke name="setApplication_" inputVariable="setApplicationReq"
operation="setApplication" partnerLink="agilpro" portType="agi:AgilproIssuer">
  <correlations>
    <correlation pattern="out" set="atmInteraction"/>
  </correlations>
</invoke>
<assign name="startAction_Readytowork" >
  <copy>
    <from part="Ticketnumber" variable="nextActionReq"/>
    <to part="Ticketnumber" variable="startActionReq"/>
  </copy>
</assign>
<invoke name="startAction_Readytowork"
inputVariable="startActionReq" operation="startAction" partnerLink="agilpro"
portType="agi:AgilproIssuer">
  <correlations>
    <correlation pattern="out" set="atmInteraction"/>
  </correlations>
</invoke>
<receive variable="nextActionReq" operation="nextAction"
partnerLink="atm" portType="atm:FrontEnd">
  <correlations>
    <correlation set="atmInteraction"/>
  </correlations>
</receive>
<assign name="reply_Readytowork">
  <copy>
    <from expression="string('Readytowork')"/>
    <to part="nextActionOUT" variable="nextActionRes"/>
  </copy>
</assign>
<reply operation="nextAction" partnerLink="atm"
portType="atm:FrontEnd" variable="nextActionRes"/>
<assign name="endAction_Readytowork">
  <copy>
    <from part="Ticketnumber" variable="nextActionReq"/>
    <to part="Ticketnumber" variable="endActionReq"/>
  </copy>
</assign>
<invoke name="endAction_Readytowork" inputVariable="endActionReq"
operation="endAction" partnerLink="agilpro" portType="agi:AgilproIssuer">
  <correlations>
    <correlation pattern="out" set="atmInteraction"/>
  </correlations>

```

```
        </invoke>
      </sequence>
    </scope>
  </sequence>
  <assign>
    <copy>
      <from part="Ticketnumber" variable="nextActionReq"/>
      <to part="Ticketnumber" variable="disconnectAgilproReq"/>
    </copy>
  </assign>
  <invoke inputVariable="disconnectAgilproReq" name="disconnectAgilpro"
operation="disconnectAgilpro" outputVariable="disconnectAgilproRes"
partnerLink="agilpro" portType="agi:AgilproIssuer">
    <correlations>
      <correlation pattern="out" set="atmInteraction"/>
    </correlations>
  </invoke>
</sequence>
</scope>
</sequence>
</process>
```

## B) JWT2BPEL DIAGRAM

Using NetBeans to get a graphical representation of the BPEL process presented in the previous appendix, we get the following. The left diagram represents the whole process, with the central part collapsed (the node with a '+' symbol). If we expand this node, we get the right diagram. On that right diagram, we can see two nodes with a '+' symbol inside. They are respectively the 'Employee Arrival' activity and the 'Ready to work' activity of the BPMN original diagram. From the 'Employee Arrival' node, we see a BPEL scope labeled C5 from which events are attached. Those events are seen on its right side. Those events are used to implement in BPEL the parallel and unstructured flow specified in the original BPMN diagram.



## BIBLIOGRAPHY

- [1] OASIS, The BPEL Specification; (2007) <http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf>
- [2] OMG, The BPMN Specification; (2006) <http://www.bpmn.org/Documents/OMG%20Final%20Adopted%20BPMN%201-0%20Spec%2006-02-01.pdf>
- [3] Bohm, Corrado and Giuseppe Jacopini, Flow diagrams, Turing machines and languages with only two formation rules; (1966)
- [4] Dijkstra, Edsger, Go To Statement Considered Harmful; (1968)  
<http://www.acm.org/classics/oct95/>
- [5] B. Kiepuszewski and A. H. M. Ter Hofstede and C. Bussler, On Structured Workflow Modelling; (2000) <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.85.2336>
- [6] Milner, R. 1989 Communication and Concurrency. Prentice-Hall, Inc.
- [7] Eike Best and Raymond Devillers and Astrid Kiehn and Lucia Pomello, Concurrent bisimulations in Petri nets; (1991)
- [8] Chun Ouyang and Marlon Dumas and Arthur H. M. Ter Hofstede, From Business Process Models to Process-oriented Software Systems: The BPMN to BPEL Way; (2006)  
<http://is.tm.tue.nl/staff/wvdaalst/BPMcenter/reports/2006/BPM-06-27.pdf>
- [9] Wil M.P. van der Aalst<sup>1,2</sup> and Kristian Bisgaard Lassen<sup>2</sup>, Translating Workflow Nets to BPEL; () [http://fp.tm.tue.nl/beta/publications/working\\_papers/Beta\\_wp145.pdf](http://fp.tm.tue.nl/beta/publications/working_papers/Beta_wp145.pdf)
- [10] Stephen A. White, Process Modeling Notations and Workflow Patterns; ()  
[http://www.bpmn.org/Documents/Notations and Workflow Patterns.pdf](http://www.bpmn.org/Documents/Notations_and_Workflow_Patterns.pdf)