PROCE: an agent-based PROcess Composition and execution Environment

Francesco De Luca
Herzum Software Solution Center – Srl
87036, c.da Lecco, Rende (CS), Italy
fdeluca@herzumsoftware.com

Andrea Tundis*, Alfredo Garro
Dipartimento di Elettronica, Informatica e Sistemistica (DEIS)
Università della Calabria
87036, Arcavacata di Rende (CS), Italy
{atundis, garro}@deis.unical.it

Abstract—**The paper presents PROCE (PROcess Composition and execution Environment), an agent-based CAME and CASE tool which supports the composition of software development processes, based on the Method Engineering paradigm, and their execution. In particular, the current release of PROCE provides an integrated and flexible environment for the design of SOA applications.**

Keywords-Multi Agent System; Method Engineering; Method Fragment; Service Oriented Architecture; Services Composition.

I. Introduction

The development of complex software applications can be supported by Software Engineering Processes (SEPs) which, according to the Method Engineering (ME) paradigm, are obtained by composing a set of purposely selected method fragments [6]. This approach allows combining the need of defining specifically tailored methodologies with the possibility of reusing existing methodologies (or their portions) well documented and experimented. However, the concrete use of the ME paradigm requires the availability of suitable models and techniques to represent method fragments and their composition. In addition, a central issue is represented by the availability of Computer-Aided Method Engineering (CAME) tool, to support the composition of development processes through discovering and assembling of method fragments, and of Computer-Aided Software Engineering (CASE) tool, to support the execution of obtained processes.

In this context, the paper presents PROCE (PROcess Composition and execution Environment) which represents both a CAME and a CASE tool and then is capable to support both the definition of a development process, by the selection and composition of method fragments, and its subsequent execution.

PROCE is a Multi Agent System, in which both method fragments and the processes derived from their composition are *represented* by agents. This approach provides an effective solution to the issue of fragment composition that can be based on the cooperation among agents. Moreover, the agent-based representation of the development process allows a more effective process execution. In fact, the *agentified* method fragments, in the CAME phase, cooperate to build up the development process, whereas, in the CASE stage, cooperate to support its execution (each agent is in charge of a portion of the process and interacts with the others by exchanging fragment work products).

In order to verify the effectiveness and the efficacy of PROCE in method fragments composition and processes execution, a preliminary experimentation was carried out in the Service Oriented Architecture (SOA) domain [1]. In fact, despite its popularity, the development of SOA applications is not well supported by methodologies and tools that easily adapt to the needs of specific applications to be implemented; as a consequence, the efforts required for adapting an existing methodology often makes profitable to define a new one [10]. Therefore, also in the SOA domain, the ME approach can provide, as for the Object Oriented (OO) [6] and Agent Oriented (AO) domains [2], an effective solution able to combine the definition of *ad-hoc* methodology with the reuse of existing ones.

The paper is organized as follows: Section II and III present system requirements and design respectively. The system implementation details are presented in Section IV, whereas Section V reports an application example. Finally, conclusions are drawn and future works delineated.

II. System Requirements

PROCE should provide both CAME and CASE features so the following main requirements have been identified:

1. *Method Base management*: the tool should be able to access and manage a repository (Method Base) in which method fragments are stored and collected.
2. *Process definition and verification:* the tool should allow the selection and Work Products (WPs) based composition of the available method fragments and should provide techniques for checking the feasibility of the obtained development process.
3. *Process execution*: the tool should be able to instantiate and support the execution of a process obtained from method fragments composition.

III. System Design

PROCE has been designed following the *Organization-based Multi Agent System Engineering* (O-MASE) methodology [4] and the fragment definition provided by the IEEE FIPA specifications [3]. In particular, the logical architecture of PROCE and its behavior are described in Section A and B respectively.

**corresponding author*

A. *High Level architecture*

PROCE is organized as a cooperative society of agents each of which covers at least one *Role*. A *Role* represents a relationship between a *Goal* and the *Capability* used by a software agent to achieve it. The following *Roles* have been identified: (i) *Process Builder Role* who has in charge the definition of specific Software Engineering Processes (SEPs); (ii) *Process Coordinator Role* who has in charge the execution of a specific development process; (iii) *Method Fragment Role* who has in charge the execution of a specific method fragment and the management of its WPs (Figure 1).

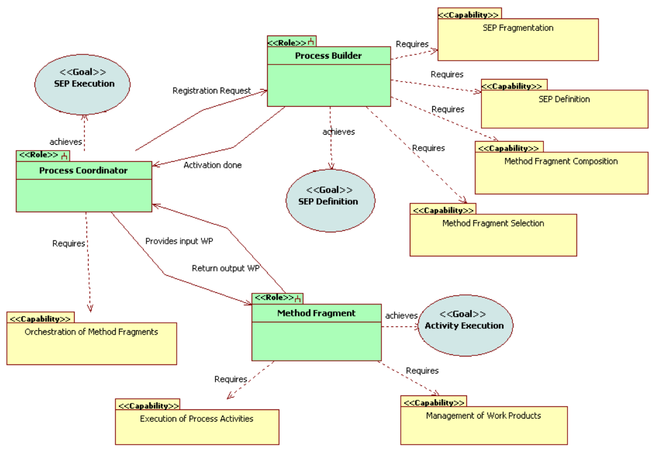


Figure 1. Logical Architecture

B. *System behavior*

System behavior depends on the *Roles* played by the component *Agents* and their interactions.

In particular, two main phases can be identified: *Process Composition* and *Process Execution*. In the first phase, the system supports the *ad-hoc* definition of a new process through the composition of method fragments available in a Method Base. In particular, an *Agent* which perform the *Process Builder Role*, interact with the user for the definition of the specification of the process under construction. According to SPEM [8], the process is specified in terms of the component *Activities*, their relationships and the exchanged WPs. Starting from this process specification, the *Builder* interacts with the agents which perform the *Method Fragment Role*, requiring them, by exploiting a *contract-net* based protocol [5], to cover specific *Activities* of the process. If this *fragments negotiation* phase successfully completes, an *Agent* which performs the *Process Coordinator Role* is instantiated by specifying the process to coordinate in terms of the involved *Fragments*.

In the *Process Execution* phase *Agents* which perform the *Method Fragment Role* are in charge of executing the associated *Activities* of the development process. These Agents are orchestrated by the Agent which performs the *Process Coordinator Role* who drives the exchange of WPs among the involved Agents.

IV. SYSTEM IMPLEMENTATION

PROCE has been currently implemented as a stand-alone Java application. The development of the CAME features is in progress whereas the CASE features have been fully implemented. Moreover, due to the increasing interested in the SOA domain, the current experimentation has involved development processes and method fragments related to the development of service-oriented applications as it requires addressing several issues ranging from the definition of the application to the discovery, development, composition, integration and testing of SOA services.

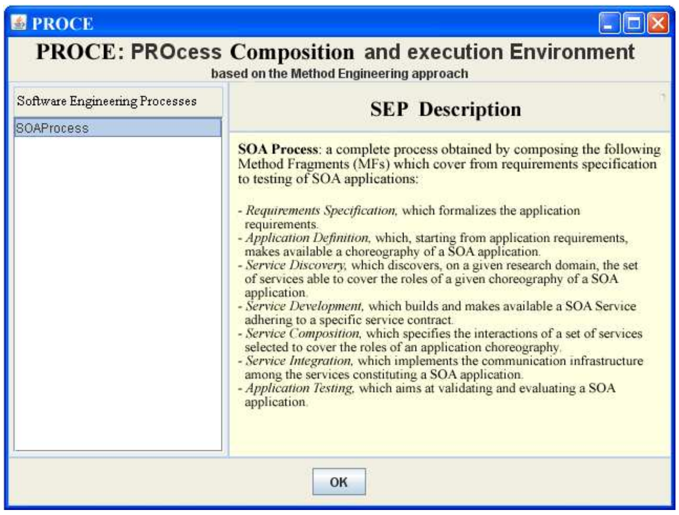
In particular, the following agents have been implemented: (i) *SOA Process Agent* covering and extending the *Process Coordinator Role* to support the execution of a SEP for the development of SOA applications; (iii) several *Method Fragment (MF) Agents* that cover and extend the *Fragment Role* to support the execution of specific portions of the above introduced SOA process.

As an example, the *MF-Service Composer Agent* has been associated to a MF which, starting from an *Application Choreography*, represented by a WS-CDL (Web Services Choreography Description Language) document, and from a *mapping* among *roleTypes* of the *Choreography* and real *Web Services*, produces a WS-BPEL (Web Services Business Process Execution Language) document specifying the orchestration among the involved services in terms of data and messages exchanged and task execution sequences [11].

PROCE has been developed by adopting various platform-independent technologies and open standards to ensure interoperability; in particular JADE (Java Agent Development Framework), an agent-based framework that allows the development of MAS and provides a runtime execution platform, has been exploited [7].

V. USING PROCE

Figure 2 shows the PROCE GUI; in the left column the list of the available SEPs is reported; currently only the *SOAProcess* is available which is briefly described on the right side.


PROCE

PROCE: PROcess Composition and execution Environment
based on the Method Engineering approach

Software Engineering Processes

SOAProcess

SEP Description

SOA Process: a complete process obtained by composing the following Method Fragments (MFs) which cover from requirements specification to testing of SOA applications:

- *Requirements Specification*, which formalizes the application requirements.
- *Application Definition*, which, starting from application requirements, makes available a choreography of a SOA application.
- *Service Discovery*, which discovers, on a given research domain, the set of services able to cover the roles of a given choreography of a SOA application.
- *Service Development*, which builds and makes available a SOA Service adhering to a specific service contract.
- *Service Composition*, which specifies the interactions of a set of services selected to cover the roles of an application choreography.
- *Service Integration*, which implements the communication infrastructure among the services constituting a SOA application.
- *Application Testing*, which aims at validating and evaluating a SOA application.

Figure 2. The PROCE Graphical User Interface (GUI)

By selecting the available *SOAProcess* a SPEM diagram of the process is shown; in the diagram the *Activities* composing the selected process and their work products are reported (see

Figure 3). Each Activity is associated to a Method Fragment which can be then executed. Before starting the execution of a MF, the *SOAProcess Agent* verifies that the required input WPs are available and in the format expected by the related *MF Agent*. In this case, the *SOAProcess Agent* provides the input WPs to the *MF Agent*, on the contrary it informs the *MF Agent* so that it can ask the user for the required input WPs.

In the following, the execution of the *Services Composition* MF carried on by the *MF-Service Composer Agent* is shown.

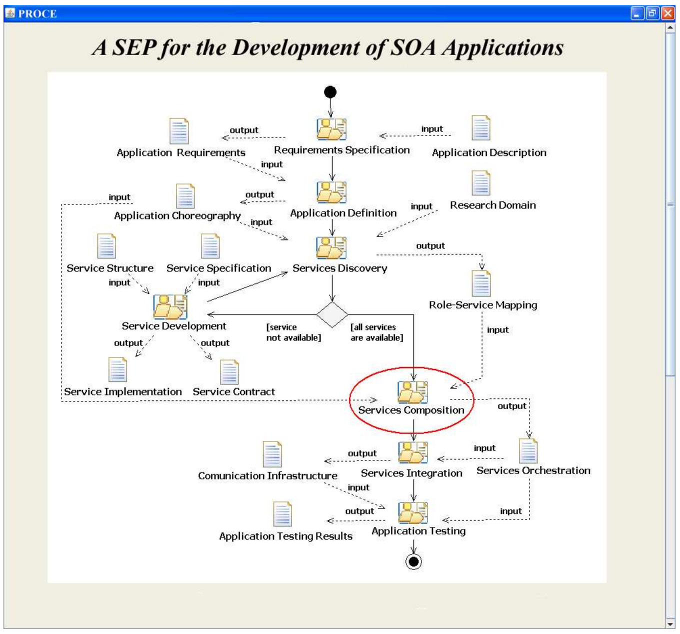
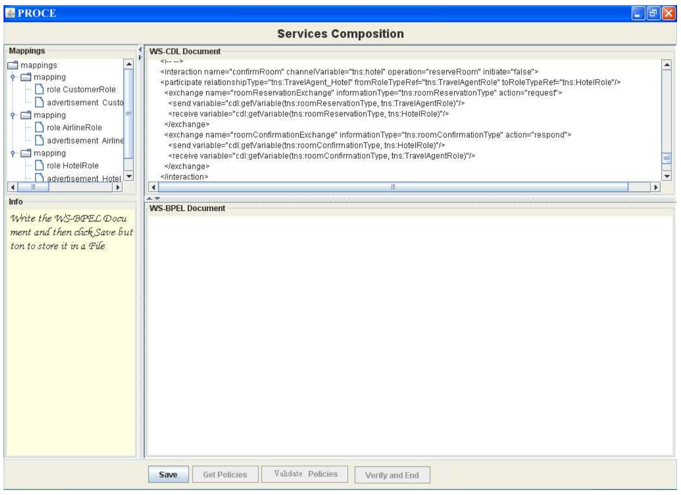


Figure 3. A SEP for the development of SOA Applications

In this first phase of the MF execution, the *MF-Service Composer Agent* obtains the two input WPs (see Figure 3); in particular, the provided *Role-Service Mapping* and the *Application Choreography* are represented on the upper-left and on the upper-right side of Figure 4 respectively.


PROCE

Services Composition

Mappings
- mappings
  - mapping
    - role CustomerRole
    - advertisement Custo
  - mapping
    - role AirlineRole
    - advertisement Airline
  - mapping
    - role HotelRole
    - advertisement Hotel

WS-CDL Document

```
<!-- -->
<interaction name="confirmRoom" channelVariable="tns:hotel" operation="reserveRoom" initiate="false">
<participate relationshipType="tns:TravelAgent_Hotel" fromRoleTypeRef="tns:TravelAgentRole" toRoleTypeRef="tns:HotelRole"/>
 <exchange name="roomReservationExchange" informationType="tns:roomReservationType" action="request">
  <send variable="cdl:getVariable(tns:roomReservationType, tns:TravelAgentRole)"/>
  <receive variable="cdl:getVariable(tns:roomReservationType, tns:HotelRole)"/>
 </exchange>
 <exchange name="roomConfirmationExchange" informationType="tns:roomConfirmationType" action="respond">
  <send variable="cdl:getVariable(tns:roomConfirmationType, tns:HotelRole)"/>
  <receive variable="cdl:getVariable(tns:roomConfirmationType, tns:TravelAgentRole)"/>
 </exchange>
</interaction>
```

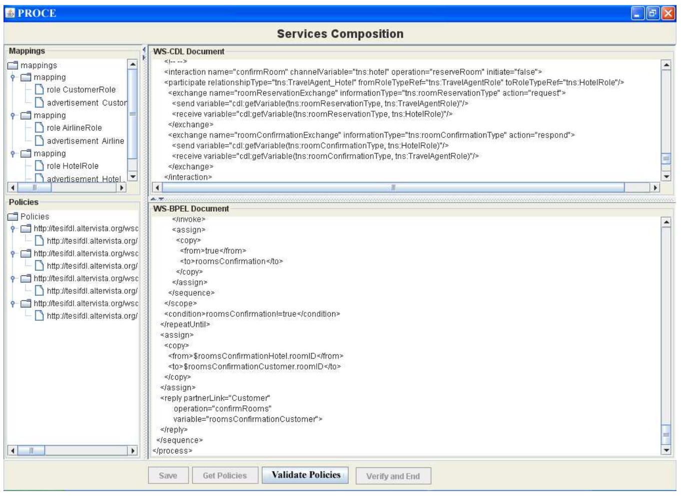
Info

Write the WS-BPEL Document and then click Save button to store it in a File

WS-BPEL Document

Figure 4. The Services Compositon Environment

In the next phase, for each *roleType* in the *Choreography* a specific Web Service is selected among those available in the mapping. As the selected Services can be managed by different SOA domains, it is necessary to check and validate their policies; thus, the *MF Service Composer Agent* supports the user in getting (*Get Policies* button) the policies associated to each service and validating them (*Validate Policies* button).


PROCE

Services Composition

Mappings
- mappings
  - mapping
    - role CustomerRole
    - advertisement Custor
  - mapping
    - role AirlineRole
    - advertisement Airline
  - mapping
    - role HotelRole
    - advertisement Hotel

WS-CDL Document

```
<!-- -->
<interaction name="confirmRoom" channelVariable="tns:hotel" operation="reserveRoom" initiate="false">
<participate relationshipType="tns:TravelAgent_Hotel" fromRoleTypeRef="tns:TravelAgentRole" toRoleTypeRef="tns:HotelRole"/>
 <exchange name="roomReservationExchange" informationType="tns:roomReservationType" action="request">
  <send variable="cdl:getVariable(tns:roomReservationType, tns:TravelAgentRole)"/>
  <receive variable="cdl:getVariable(tns:roomReservationType, tns:HotelRole)"/>
 </exchange>
 <exchange name="roomConfirmationExchange" informationType="tns:roomConfirmationType" action="respond">
  <send variable="cdl:getVariable(tns:roomConfirmationType, tns:HotelRole)"/>
  <receive variable="cdl:getVariable(tns:roomConfirmationType, tns:TravelAgentRole)"/>
 </exchange>
</interaction>
```

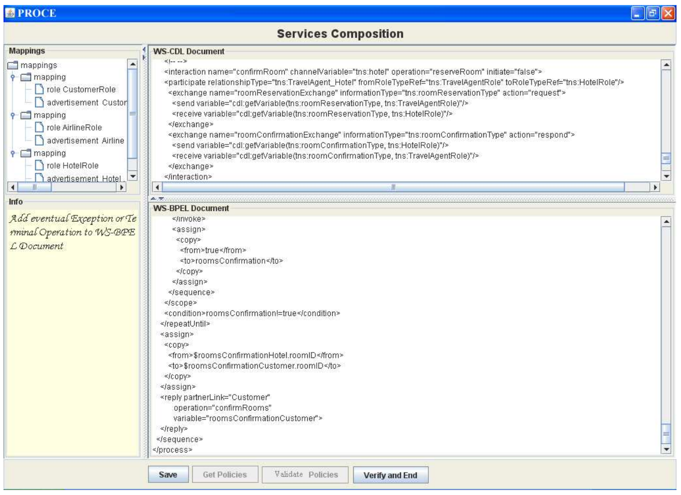
Policies
- Policies
  - http://tesifdl.altervista.org/wsc
    - http://tesifdl.altervista.org/
  - http://tesifdl.altervista.org/wsc
    - http://tesifdl.altervista.org/
  - http://tesifdl.altervista.org/wsc
    - http://tesifdl.altervista.org/
  - http://tesifdl.altervista.org/wsc
    - http://tesifdl.altervista.org/

WS-BPEL Document

```
      </invoke>
      <assign>
       <copy>
        <from>true</from>
        <to>roomsConfirmation</to>
       </copy>
      </assign>
     </sequence>
    </scope>
    <condition>roomsConfirmation!=true</condition>
   </repeatUntil>
   <assign>
    <copy>
     <from>$roomsConfirmationHotel.roomID</from>
     <to>$roomsConfirmationCustomer.roomID</to>
    </copy>
   </assign>
   <reply partnerLink="Customer"
       operation="confirmRooms"
       variable="roomsConfirmationCustomer">
   </reply>
  </sequence>
</process>
```

Figure 5. Management of Services Policies

Finally, Figure 6 shows the last phase of the execution of the *Services Composition* MF, in which a WS-BPEL is produced (*Verify and End* button) after verifying that the selected services correctly cover all the roles involved in the *Application Choreography*.


PROCE

Services Composition

Mappings
- mappings
  - mapping
    - role CustomerRole
    - advertisement Custor
  - mapping
    - role AirlineRole
    - advertisement Airline
  - mapping
    - role HotelRole
    - advertisement Hotel

WS-CDL Document

```
<!-- -->
<interaction name="confirmRoom" channelVariable="tns:hotel" operation="reserveRoom" initiate="false">
<participate relationshipType="tns:TravelAgent_Hotel" fromRoleTypeRef="tns:TravelAgentRole" toRoleTypeRef="tns:HotelRole"/>
 <exchange name="roomReservationExchange" informationType="tns:roomReservationType" action="request">
  <send variable="cdl:getVariable(tns:roomReservationType, tns:TravelAgentRole)"/>
  <receive variable="cdl:getVariable(tns:roomReservationType, tns:HotelRole)"/>
 </exchange>
 <exchange name="roomConfirmationExchange" informationType="tns:roomConfirmationType" action="respond">
  <send variable="cdl:getVariable(tns:roomConfirmationType, tns:HotelRole)"/>
  <receive variable="cdl:getVariable(tns:roomConfirmationType, tns:TravelAgentRole)"/>
 </exchange>
</interaction>
```

Info

Add eventual Exception or Terminal Operation to WS-BPEL Document

WS-BPEL Document

```
      </invoke>
      <assign>
       <copy>
        <from>true</from>
        <to>roomsConfirmation</to>
       </copy>
      </assign>
     </sequence>
    </scope>
    <condition>roomsConfirmation!=true</condition>
   </repeatUntil>
   <assign>
    <copy>
     <from>$roomsConfirmationHotel.roomID</from>
     <to>$roomsConfirmationCustomer.roomID</to>
    </copy>
   </assign>
   <reply partnerLink="Customer"
       operation="confirmRooms"
       variable="roomsConfirmationCustomer">
   </reply>
  </sequence>
</process>
```

Figure 6. Services Orchestration

VI. CONCLUSIONS AND FUTURE WORK

PROCE, a MAS for method fragments composition and execution of software development processes, has been presented along with an application example in the SOA domain showing its effectiveness and user-orientation. The current release of PROCE is available under the LGPL license on the Web Site of the *OpenKnowTech* project [9].

Future efforts are geared to: (i) the definition of new method fragments concerning the different aspects of the development of SOA applications; (ii) the implementation of the CAME features of the tool ; (iii) the implementation of the future release of PROCE as an Eclipse plug-in so to benefit from its popularity and supporting community.

REFERENCES

[1] Bruni, R., Lluch Lafuente, A., Montanari, U., Tuosto, E.: Service Oriented Architectural Design. In: Proceedings of the 3rd International Symposium on Trustworthy Global Computing (TGC'07). LNCS, vol. 4912, pp. 186--203, Springer, Heidelberg (2008).

[2] Cossentino, M., Fortino, G., Garro, A., Mascillaro, S., Russo, W.: PASSIM: a simulation-based process for the development of multi-agent systems. Int. J. of Agent-Oriented Software Engineering, vol. 2, n.2, pp. 132--170, Inderscience Enterprises Ltd., United Kingdom (2008).

[3] Cossentino, M., Gaglio, S., Garro, A., Seidita, V.: Method fragments for agent design methodologies: from standardisation to research. Int. J. of Agent-Oriented Software Engineering, vol. 1, n. 1, pp. 91--121, Inderscience Enterprises Ltd (2007).

[4] Deloach, S. A., Gracia-Ojieda, J. C., Oyenan, W. H., Valenzuela, J.: O-MaSE: A Customizable Approach to Developing Multiagent Development Processes. In the 8th International Workshop on Agent Oriented Software Engineering, Honolulu, USA (2007).

[5] FIPA - Contract Net Interaction Protocol Specification - http://www.fipa.org/specs/fipa00029/.

[6] Henderson-Sellers, B.: Method engineering for OO systems development. Communications of the ACM, vol. 46, n. 10, pp. 73--78, ACM press (2003).

[7] JADE - Java Agent Development Framework - http://jade.tilab.com/

[8] OMG SPEM - Systems Process Engineering Metamodel Specification - http://www.omg.org/spec/SPEM/2.0/

[9] OpenKnowTech Project - http://www.openknowtech.it/

[10] Ramollari, E., Dranidis, D., Simons, A.J.H.: A survey of service-oriented development methodologies. In: Proceedings of the 2nd Young Researchers' Workshop on Service Oriented Computing, Leicester, UK, pp. 75--80 (2007).

[11] W3C Web of Services, standards and technologies, http://www.w3.org/standards/