Intelligent business processes composition based on multi-agent systems

José A. García Coria[a], José A. Castellanos-Garzón[b], Juan M. Corchado[c]*

Department of Computers and Automation, University of Salamanca, Plaza de la Merced s/n, 37008 Salamanca, Spain

Abstract

This paper proposes a novel model for automatic construction of business processes called IPCASCI (*Intelligent business Processes Composition based on multi-Agent systems, Semantics and Cloud Integration*). The software development industry requires agile construction of new products able to adapt to the emerging needs of a changing market. In this context, we present a method of software component reuse as a model (or methodology), which facilitates the semi-automatic reuse of *web* services on a *cloud computing* environment, leading to business process composition. The proposal is based on *web* service technology, including: (i) Automatic discovery of *web* services; (ii) Semantics description of *web* services; (iii) Automatic composition of existing *web* services to generate new ones; (iv) Automatic invocation of *web* services. As a result of this proposal, we have presented its implementation (as a tool) on a real case study. The evaluation of the case study and its results are proof of the reliability of IPCASCI.

Keywords: Multi-agent system, Web service, Cloud computing, Software component reuse, Software engineering, Business process composition, Machine learning.

* Corresponding author. *E-mail address:* {jalberto[a], jantonio[b], corchado[c]}@usal.es

1. Introduction

The software development industry is constantly evolving and looking for new technologies, languages and tools that are increasingly powerful, efficient and safe. In this process, it is essential to build models, architectures and agile technologies able to introduce new tools as simply and economically as possible. Component reuse is actually one of the techniques that more clearly contribute to such a development by providing efficient mechanisms to create quality software [1-6]. Reuse increases software reliability (because it uses tested software components), development productivity and implies a clear cost reduction [6-8]. Since the recent increasing of the volume and complexity of software products, reuse has been a field taken into great consideration, being a fundamental stage in design and quality models as CMMI (Capability Maturity Model Integration) [9].

In this context, *web* service reuse appears as an interesting alternative with respect to code classical reuse. Indeed, a *web* service is a software component representing a service deployed on a *web* platform and supporting automatic interactions between machines on a computer network [10]. In this framework, arises SOA (Service-Oriented Architecture) as a new architecture leading to conventional software development [11-13]. SOA introduces a new method to create distributed applications where their basic services can be discovered, published, and linked in order to achieve more complex services. Applications interact through the existing services from entry points on an interface rather than at the level of implementation [14].

Software development can be analyzed from different perspectives, offering us a wide variety of alternatives to a methodological, functional and instrumental level, among others. One of the paradigms revolutionizing this industry in recent years has been *cloud computing* [15-19]. The *cloud* represents a novel concept of service and information distribution, providing many possibilities of scaling solutions, facilitating the use of relatively simple and economic terminals (interesting models of pay per use, and multi-platform accessibility, etc). However, this model has certain drawbacks related to maintenance complexity and application development [20, 21]. Moreover, the number of experienced engineers in this field is relatively low and the development time is significantly high [22-24].

Starting from all the above, this research proposes a methodology (IPCASCI, *Intelligent business Processes Composition based on multi-Agent systems, Semantics and Cloud Integration*) that facilitates the business process construction on *cloud computing* environments in an agile and efficient way from developed components. It deals with the development of a proposal that facilitates the creation of such processes in form of *web* services from other semi-automatic functional services. All this is carried out on the framework of a computer system guided by relatively inexpert programmers. The process of business process construction is guided by a multi-agent architecture based on virtual organizations [25-27], which is able to implement the intelligent behavior needed for process management by using *ontology* [28-32]. The goal of this research is then to provide a model allowing the automatic construction of a business process from specifications in text format (with some constraints) of the process concerning us. The multi-agent system based on virtual organizations will facilitate the process composition by using standard BPEL (Business Process Execution Language). This standard allows the automatic composition of *web* services in an easy way by adding the advantage of a direct projection to a diagram BPMN (Business Process Management Notation) [33].

To reach the goals proposed in this research, the remainder of the present paper has been structured into the following sections: Section 2 outlines information and works related to this research. Section 3 deals with an overview of the different components coupled to our proposal to make the *web* service composition, that is, architecture IPCASCI. This section gives a general idea of the whole composition process from IPCASCI. Section 4 presents *cloud computing* and *web* service technology, which have been included in our proposal. On one hand, we describe the advantages of developing IPCASCI on a *cloud* environment and provide two *cloud* functionalities to support its performance. On the other hand, we introduce *web* services technology and explain the inclusion of semantics to the *web* services through *ontology* to later facilitate the process of discovery and composition from a multi-agent system. Section 5 describes the agent-based virtual organizations defined in the multi-agent system of IPCASCI to make the discovery of the *web* services fit the specifications of the user to compose them and obtain the solution *web* service. Section 6 develops a real case

study to give an implementation of IPCASCI, which is evaluated through software metrics. Conclusions, Appendix A with information supporting our proposal and the references of this research have been given at the end of this paper.

2. Related work

As previously explained, we have introduced an architecture (IPCASCI) aimed at composing *web* services to obtain new ones in an automatic way by starting from requirements given by the user. This process is carried out in such a way that the service automatic construction be efficient and has an intelligent behavior. Intelligent behavior is intended as the ability of processing of the input requirements introduced by the user, that is, the system be able to: (i) analyze the input, (ii) find the *web* services allowing to hold the requirements, and (iii) carry out an automatic composition of *web* services for their corresponding business processes. Thus, as a final result, new *web* services implementing the provided requirements will be achieved in an automatic process of discovery and composition. In this sense, there exist several approximations of architectures based on semantic *web* services. In [34], a solution focused on *fuzzy logic* to discover semantic *web* services has been proposed. In [35], a solution based on agents and ontological language DAML-S has been proposed, and in [36], a proposal based on queries SPARQL and ontological language OWL-S has also been used. In general terms, most of the proposals on architectures of semantic *web* services are based on language OWL-S. The proposal given in this paper differs from the existing ones in the following:

- Design of a global platform, which is embedded in a *cloud* environment and whose structure has been provided for an agile and efficient running.
- The semantics information of the *web* services does not depend on the internal structure of the ontology, which allows reusing ontologies regardless of their formats.
- We have included a *multi-agent system* based on virtual organizations that facilitates requirement analysis of the user and the discovery process of *web* services.

- The solution *web* service has been built from a simple definition given through a set of specifications introduced by the client.

3. Architecture IPCASCI for business process composition

This section describes the different components integrated to our proposal IPCASCI, which allow carrying out the automatic composition of *web* services. So Figure 1 shows a conceptual diagram of the components integrating the proposed architecture. This figure describes the following items:

1. *Cloud system:* Platform of cloud computing on which is supported our proposal. The platform provides an environment for running and storage.
2. *Web services:* The *web* services existing in the architecture that will be used by the process of *web* service composition.
3. *UDDI register:* Universal register system where used *web* services are registered. This allows us an open access to the *web* services of the architecture. This way, the *web* services implemented on the architecture may be reused regardless of our proposal.
4. *Multi-agent system based on virtual organization:* This system supports functionalities to make the discovery and the composition of *web* services:
    a. *Analysis system:* Analyze the semantic content introduced by the user in order to structure it in computable items.
    b. *Search system:* Discover the *web* services that hold the semantics as syntactic constraints given by the user.
    c. *Composition system:* Once determined the *web* services and their relationships holding the user semantic requirements, a service composition is applied on such *web* services to obtain the new *web* service.
5. *Ontologies:* Distinct ontologies modeling the semantic knowledge that can be included in the *web* services.
6. *BPEL file:* Store the composition of the *web* services that meet the requirements indicated by the user to obtain the desired solution.

**** Figure 1 here ****

As a general schema of working, IPCASCI accepts the requirements of the user through the assistant software to raise the processes of discovery and composition. The steps followed by IPCASCI have been given the following algorithm and represented in Figure 2:

3.1. Algorithm for web service discovery and composition (AWSDC).

1. The user introduces the requirements of the *web* service to build by means of an assisted system. Such a system allows users to define a list of terms (in form of modules) representing the requirements of the new *web* service. The list of terms is bounded by the *web* service repository in the *cloud* system. The result of this process is a set of related modules (*analysis system*) in such a way that each module registers the following information:
   a. Name;
   b. Input;
   c. Output;
   d. Process carried out by the module (semantic concept it represents);
   e. Domain of the semantic concept represented by the module (ontological domain);
   f. Preconditions of running;
   g. Relationship with other modules.
2. For each module of the above step, an automatic search (*search system*) for the *web* services that include the requirements of the module is carried out. Moreover, the search is made in syntactic and semantics form.
3. From the information of the above step, a reduced BPD diagram (Business Process Diagram, [37]) is built through of graphic standard BPMN (Business Process Management Notation), on which each *web* service is represented as an activity and also showing the interactions existing between the *web* services.

4. The BPD diagram is then displayed by the assistant software to the user, because could exist several *web* services implementing a single activity. In this case, the user can select another *web* service different from the one given by default in the diagram.
5. From the BPD diagram, a composition BPEL (Business Process Execution Language, [38]) is carried out in such a way that the service specified by the user is built and can later be invoked (*composition system*) as one more service in the platform. See Figure 2 and 3 to summarize the whole process.

As shown in Figure 3, the assistant software interprets the requirements introduced by the user (the requirements are restricted to the ontological domain defined on the addressed problem) to build the new *web* service. These requirements are later processed by IPCASCI in the *cloud* system, which runs a multi-agent system based on virtual organizations to discover and compose the *web* services including the requirements of the user. As a result, a BPMN (Business Process Management Notation) solution is obtained which represents, through activities, the functionalities of the required *web* service. Such a solution can be fixed by the user to confirm the diagram. Finally, the new *web* service is generated and included in the platform, so that users/clients can use it.

**** Figure 2 here ****

**** Figure 3 here ****

Note that with respect to standards BPMN and BPEL, we have that the first one specifies a diagram of business process (BPD), offering several advantages [37] as modeling business processes in a simple way, to be readable by nontechnical users, providing tools to model complex business processes and simple mapping to business execution languages as BPEL. This way, BPEL is the standard to provide business process specifications for a *web* service

environment. It allows composing *web* services, representing them as an operation flow in a diagram, and states such a composition as a new *web* service.

4. Cloud computing and *web* services in IPCASCI

This section describes the functionalities provided by the *cloud* system and *web* service technology coupled to architecture IPCASCI. The idea is to give a more detailed description of the associated points of the algorithm presented in the above section needing further background.

4.1. Cloud system

We have defined an architecture for *web* service composition on a *cloud computing* environment, which implies that the software generated from it will be of shared and distributed use. Moreover, its management should be efficient, reliable and the access to it should be from any platform [22, 24, 39]. These characteristics provide the development of scalable, dynamic and distributed software, representing advantages on the current distributed systems [40]. All this, allows us integrates components as a multi-agent system and SOA (*service-oriented architecture,* [11-13]) to our architecture [41, 42].

According to the above, the services offered by the proposed model through *cloud computing* are focused on the layers of *infrastructure* and *software*. The infrastructure layer provides services as file storage, which relieves the client for the liability of acquiring the corresponding storage systems and maintenance. On the other hand, the software layer provides a set of services REST [43], allowing managing operations for automatic discovery and composition of *web* services. Particularly, the proposed model provides two functionalities, namely: *analyze&Discover* and *composeService.* The first functionality accepts as input a XML document (Extensible Markup Language, [44]) representing the requirements of the user for which will be generated a diagram according to standard BPMN that models the input, Figure 4. For each activity of the diagram, there is a set of *web* services fixed the semantic and syntactic requirements of the concept

represented by the activity (module/concept introduced by the user). Therefore, the output of this functionality is another XML document representing a mapping of the input to a BPMN graphic. The second functionality accepts as an input a diagram BPMN in a XML document, which is a reduced diagram, that is, each activity of the diagram is associated with a single *web* service. From this input, the composition of the *web* services associated with each activity will be carried out to finally build the required *web* service by means of BPEL (Business Process Execution Language). Note that both functionalities encapsulate the multi-agent system for discovering *web* services and the composition of such services, respectively. This allows us an abstraction level from the different components coupled to the architecture from the implementation point of view.

*** *Figure 4 here* ***

4.2. Web services and inclusion of semantics

The main goal of our proposal is *web* service reuse for its composition in order to generate new *web* services, automatically. Starting from this, we have focused on SOA technology (Service Oriented Architecture, [45]) to represent *web* services by means of description WSDL, register UDDI, messaging common protocols (SOAP). The choosing of *web* service reuse is due to its modular nature uncoupling the interface of the service from the implementation, and the ability of linking dynamically services.

As previously explained, the engine of *web* service discovery and analysis has been given by a multi-agent system (MAS), for which has been developed several works integrating it to SOA technology [46, 47]. Hence, we integrate *web* services and SOA following the diagram in Figure 5. As shown in this figure, the MAS represents the ontology concepts (modeled by an ontology defined in box *Ontologies,* [48, 49]) that will include the new *web* service, and is linked to register UDDI where the *web* services of the platform have been published. The description (WSDL files) of such *web* services has been given in box WSDL, which has been linked to box *Ontologies,* implying that each description WSDL of *web* services

includes the ontological concept it represents. Note that the inclusion of ontology to represent concepts is the basis of the discovery and analysis process of *web* services (*machine learning*). In this sense, the MAS given in Figure 5 creates an agent for each ontological concept to assist the discovery process of *web* services, which is based on semantic concepts.

Based then on the above, we have used language WSDL-S (Web Service Semantics, [50]) to make semantics annotations on descriptions WSDL of the *web* services. WSDL-S introduces an association mechanism of semantics annotations for *web* services described using WSDL. Thus, WSDL-S defines a domain ontology on *web* services. This means that the ontology provides labels to add semantics to *web* services through insertions (annotations) into files WSDL [51, 52], and includes three sub-ontologies for this purpose, that is: *service profile, service grounding and process model*. Summarizing, WSDL is the means of description for *web* services. On the basis of its descriptive ability, a mechanism to annotate abilities and requirements referenced from a semantic model (ontology WSDL-S) has been provided. Such characteristics allow the architecture to automate the discovery process of *web* services (*semantic matching* of user requirements with *web* services). Hence, we use a MAS to model the ontology, which create agents representing each ontological concept, connecting them (communication lines between agents) by equivalent concepts. In addition, each agent also builds a list of links to entries to the UDDI register, whose *web* services include the concept it represents, as shown in Figure 6.

*** *Figure 5 here* ***

*** *Figure 6 here* ***

5. Multi-agent sytem (MAS) for service discovery and composition in IPCASCI

This section outlines the structure and performance of the MAS involved in our architecture. As explained in Section 2 and now shown in Figure 7, this MAS consists of three agent-based virtual organization subsystems, which have been classified in *analysis*,

search and *composition system*. These systems have been built in form of *black box* whereby, they do not depend on other systems, and there are not communication lines between them. Hence, any commutation between them is scheduled by agent *manager*. This way, these agent-based virtual organizations can apart be modified or adapted without affecting the performance of the remaining systems in the architecture.

**** Figure 7 here ****

5.1. Analysis system

The *analysis system* manages the requirements introduced as input by the user. It processes the requirements by converting them into computable information and achieving enough semantics information to later carry out a search and composition of *web* services. This system is integrated by a main agent called *analysis coordinator*. The requirement input of the user has been made through a graphic assistant (software) that integrates each generated module. We have introduced the module concept to encapsulate the attributes representing a semantic concept, and it has been defined on the following characteristics:

1. *Functionality:* It describes the functionality to be carried out by the module and is introduced in text format by means of a list of semantic concepts associated with the module. Optionally, it is possible to include preconditions to hold by the module before the running.
2. *Domain:* From the definition of each module the assistant allows the user to choose the domain name of a list of existing domains. The domain classifies the workspace where we are going to perform the task and the ontology to use.
3. *Input:* The user determines the number of inputs of the module based on the concept it represents.
4. *Output:* By a similar way as the *input,* the user specifies the output generated by the module.

5. *Interconnection:* The assistant guides the user to define the connection of the current module with the previously defined modules. In this sense, the following instructions of flow control to connect modules have been defined to represent the information given by the user:
    a. *[Module_i] If [Condition] then [Module_j]*
    b. *[Module_i] If [Condition] then [Module_j] else [Module_k]*
    c. *[Module_i] Parallel output to [Module_j], [Module_k]*
    d. *[Module_i] Follows [Module_j]*

When the user completes the module definition and their relationships, the assistant builds a flow diagram with computable information aimed at generating the new *web* service and so the assistant stores the diagram in an XML document and passes to the next step. The assistant then invokes *cloud* functionality *analyze&Discover* (see Section 4.1), having as an input the flow diagram in XML format. Thus, this functionality runs the *analysis system* (and later runs the s*earch system*) of the architecture by passing a message to the *manager* agent to run the *analysis coordinator* agent, Figure 8. Hence, the *analysis coordinator* agent is responsible for interpreting the information of the XML document (by running a parser) to create an agent-based virtual organization representing the flow diagram saved in XML format by the assistant. To this end, each built agent by this virtual organization has been associated with a single module in the input XML document, representing in that way, the semantic concept assigned to such a module. Between the agents of this new virtual organization have also been created communication lines (conditional flow control) based on the module relationships described in the diagram given by the XML document. So that the flow diagram has been modeled as a graph, where nodes are agents, edges are communication lines and the whole graph represents the domain ontology. Then, each agent in this virtual organization manages input, output and description (preconditions and effects) for the module it represents, Figure 9. Moreover, as explained in the next subsection (*search system*), each agent will also be responsible for creating a list of links to the UDDI register for the *web* services that include the ontological concept it represents.

*** Figure 8 here ***

Completed the construction of the agent subsystem to represent the domain ontology, diagram BPD (Business Process Diagram) based on graphic standard BPMN (Business Process Management Notation) will be generated. In this case, every activity in the diagram has been associated with an agent of the ontology, and the flow control lines of the diagram have also been associated with the communication lines between the agents. All this process is intended to store diagram BPD to later convert it into a BPEL file (Business Process Execution Language), which will be run to build the expected *web* service. Basically, the transformations to convert the agent subsystem into a diagram BPD have been given in Table 1.

*** Figure 9 here ***

*** Table 1 here ***

In Section A.1 of Appendix A, an example of converting the agent subsystem into the BPD diagram has been given by basing on Table 1. As shown in Figure A2 of this example, the BPD diagram has been obtained from the agent subsystem given in Figure A1. In the next step, it would be generated an XML document (BPEL file) with the structure of the diagram in Figure A2 and the semantics information early given by the user. Note also that each activity on the BPD diagram will later be associated with the list of *web* services implementing the semantic concept (a *web* service of the list has been assigned by default) it represents by running the *search system*.

5.2. Search system

The *search system* is responsible for finding a list of *web* services from the semantic concepts and their syntaxes received from the *manager* agent. That is, a discovery process of

web services will be carried out through *semantic matching*. That is, a search for the *web* services whose semantics annotations on theirs descriptions WSDL match the ones previously defined in the modules. Thus, functionality *analyze&Discover* also runs the *search system* by passing diagram BPD (XML document) that the *analysis system* has built through the *manager* agent. The *search system* will also complete such a diagram BPD with the *web* services found. Then, the discovery process has been carried out once for each agent of the agent subsystem created by the *analysis system.* Thus, each agent in this subsystem invokes its skill *discover* to achieve a list of *web* services representing its semantics description. Figure 10 shows the agents integrating the *search system* and their connections.

As displayed in this figure, the *search system* consists of several agents responsible for reaching the following aims:

1. *Search coordinator:* It receives as input the description of a module (from the agent representing the concept associated with the module), which is provided by the *manager* agent. Therefore, the *search coordinator* agent is responsible for coordinating the whole search process of *web* services. It returns as a result, a list of *web* services that fits the input description.
2. *Semantic coordinator:* It receives the semantic concepts that most include the *web* services from the *search coordinator* agent. Then, for each concept, it sends a message to the *localizer* agent (that belongs to the *register subsystem*), which returns a list of links to register UDDI with the *web* services implementing such concepts. Note that through of this process the semantic matching is carried out from the concepts processed by the *semantic coordinator* agent and the semantics annotations previously made on files WSDL of the *web* services.
3. *Checking coordinator:* It receives the WSDL file of a *web* service and the conditions to be fulfilled by the service from the *search coordinator* agent. Then, it communicates with the *checking subsystem,* which determines whether such a *web* service holds the given syntactic requirements. Finally, it returns the checking result to the *search coordinator* agent.

*** Figure 10 here ***

Summarizing on the *search system,* we have that its first part consists of finding the *web* services associated with a semantic content. To do this, the *search coordinator* agent communicates with the *semantic coordinator* agent, who requests to the *localizer* agent (which is based on *register subsystem*), the list of links to the UDDI register for the *web* services that include the given concept. This process is done once for each provided concept. Then, when the *semantic coordinator* agent has found the *web* services associated with each concept, it carries out a filtering process to extract the services that implement all concepts. These last *web* services will be returned to the *search coordinator* agent as shown in Figure 11. Note that the inclusion of attribute *domain* in the ontological concepts classifies the *web* services into groups in such a way that the semantics search is made only on a specific group of services that match their domain, which improves the runtime of the search. Finally, the *search coordinator* agent passes the list of *web* services to the *checking coordinator* agent, which filters, through the *checking subsystem*, the ones that hold the syntactic constraints. The final list of *web* services associated with the given concept is assigned to the activity representing such a concept in diagram BPMN (BPD).

*** Figure 11 here ***

5.2.1. Register subsystem

As previously explained, the ontological information given by the user has been mapped on an agent subsystem, that is, a graph of interconnected agents. In this sense, the *localizer* agent of the *register subsystem* in Figure 10 sends a *broadcast message* asking to the agent subsystem (representing the ontology) for the *web* services that implement a determined concept. The agents that represent such a concept will respond by returning a list of *web* service links in the UDDI register. The *localizer* agent registers the received responses and returns all results.

5.2.2. Checking subsystem

The *checking subsystem* is responsible for verifying that the *web* services previously obtained hold the constraints imposed by the user, from the information given by the *checking coordinator*. This process develops three stages of checking to reach the desired filtering:

1. *Input checking:* It determines whether the input format of a *web* service fits the description given by the user.
2. *Output checking:* It determines whether the output format of a *web* service fits the description given by the user.
3. *Checking of the process:* It determines whether the exchange of SOAP messages for the *web* service is coherent with the description of the user.

5.3. Composition system

Once obtained diagram BPMN with the *web* services associated with each activity in the diagram, the composition of *web* services through the *composition system* will be run to achieve the solution *web* service by using standard WS-BPEL 2.0 [52]. The *composition system* runs in response to the called from *cloud*-functionality *composeService* given in Section 4.1. *composeService* receives as input an XML document with diagram BPMN and runs the *composition system* by passing a message to the *manager* agent.

There are several alternatives to carry out the composition process of *web* services, such as the variants explained in [53-56]. We have assumed the variant given in [56], which makes a mapping from graphic standard BPMN to language BPEL by extracting the information from the WSDL files corresponding to the *web* services associated with each activity in the diagram and the input XML document. To define the remaining properties of the WS-BPEL composition, we have used the method developed in [56], which defines the following steps:

1. Definition of the beginning of the process.
2. Definition of the end of the process.

3. Mapping of the parallel flow.
4. Synchronization of the parallel flow and
5. Mapping of loops.

Summarizing, the process of composing *web* services to obtain the required solution is carried out by the *composition system*. It receives the start request from the *manager* agent, for which the *composition coordinator* agent creates an agent for each activity (or service), Figure 12. These agents retrieve the needed information from the WSDL files of the associated *web* services and return such information to the *composition coordinator* agent, which generates the WS-BPEL by basing on the method given in [56]. Note that this process has an additional improvement since it performs on a distributed environment of *cloud computing*.

**** Figure 12 here ****

5.4. Insertion of new web services to the platform

The platform on which lies architecture IPCASCI is scalable with respect to the registered *web* services. That is, it is possible inserting and registering new *web* services in such a way that in a later running, they can be available to use. Since the whole system is included in a *cloud* environment, there is no a limitation with respect to the size of the service repository and it can grow up indefinitely. Then, to insert a new *web* service in the platform and can be localized, we must have the following items:

1. A file with description WSDL of the *web* service;
2. To be registered in the UDDI register;
3. File WSDL must have the ontological annotations corresponding to the semantic content it represents;
4. In case that the references to the semantics annotations do not appear in the ontologies defined in the platform, then a new ontology should be added and running a process to transform its content into computable one.

To insert the new *web* service in the platform, the *register subsystem* given by the *search system* is responsible for carrying out the operations allowing the localization of such a service through semantics search. Additionally, the semantics annotations inserted into the *web* service will be obtained from the WSDL file and for each annotation, the agent that implements the ontological concept will be found. The reference to such a *web* service in the UDDI register will be added to the list of services associated with the agent.

6. A case study for arquitecture IPCASCI

In this section we have given an implementation of architecture IPCASCI on a practical case study. The goal of this study is to evaluate our proposal on a real environment and in this way, to develop a practical tool representing the case study and using Algorithm AWSDC given in Section 3. In this sense, a set of parameters of such a tool has been evaluated to achieve results giving us a general overview of the reliability of architecture IPCASCI. Then, we can start by introducing the proposal of the case study in the following way:

This case study develops a tool to build a web service (in automatic way) to reserve a book through the book lending system in a library. The web service must have as input the identifier of the user that carries out the request and the required book. The task of the web service is then to check whether the user is registered in the system and its subscription has not expired. After successfully checking the above, the web service must then verify that if the book is available for loan. Moreover, it would also check whether the user has not exceeded the maximum number of books loaned by the library. If all the above conditions are held, then the loan is carried out. Otherwise, the user will be notified (for example, by e-mail) that the request has been denied along with the reason for it.

Hereinafter, we are going to show the processes included in the tool to convert the requirements of the user into modules and relationships oriented to the composition of existing *web* services to build the solution *web* service.

6.1. Requirements of the user

To implement the above case study we have developed a tool called *LibraryBookReserve* (or LBR for short) that has been implemented on the *Java* language. This tool couples an assistant for the requirement input of the user in order to process such information to later obtain the solution *web* service. Thus, the assistant allows the user to introduce the specifications through a graphic interface where possible to define the modules that will integrate the solution *web* service. Additionally, for every single module, the assistant also provides an interface to introduce its functionality (semantic concepts) and the relationship with other modules. Particularity, a module has been represented by a set of attributes, such as: *name, type, action, domain, input, output* and *interconnection with other modules.* In this sense, the modules generated from the case study have been defined in Table 2.

**** Table 2 here ****

As explained, the modules have been described by an ontology to define their behavior and actions. Therefore, for each module in the above table has been introduced its semantics as shown in Tables from A1 to A4 (Section A.2 in Appendix A). The attributes of the relationships between these modules have been given in Tables A5-A6 (Section A.2 in Appendix A). Figures A3, A4 (Section A.2 in Appendix A) and a general view in Figure 13 show definition examples of module and relationship from tool *LibraryBookReserve.* Note that we can build a flow diagram representing the information of the user from the defined modules and relationships. Thus, such a diagram has been generated and stored in an XML file by the tool assistant in order to start the analysis process.

*** Figure 13 here ***

6.2. Analysis process of the input requirements

As previously stated, the input requirements are passed (in XML format) to agent-based virtual organization *analysis system.* To do this, the tool calls functionality *analyze&Discover* (see Section 4.1), which in turn calls the *analysis system* in order to build the *agent subsystem* (Section 5.1) representing the domain ontology from the modules and relationships given as input to the functionality. As a result, a diagram BPMN (BPD) is generated by converting the agents into activities. Finally, it is achieved an XML file with the information of diagram BPMN. This diagram will later be completed by the information given from agent-based virtual organization *search system* (Section 5.2). Thus, functionality *analyze&Discover* returns the final result in an XML document.

6.3. The discovery process

The discovery process of the *web* services related to the agents of the *agent subsystem* (built by the *analysis system*) is started from *search system* and has two parts:

1. Finding the *web* services associated with the semantic concepts for each agent.
2. From the achieved *web* services, will be filtered those that meet the syntax given by the agents representing the ontological concepts (that is, holding the syntax for input, output and preconditions).

According to this, Table A7 (Section A.2 in Appendix A) shows the *web* services available in the platform of the case study, which can be used in the *web* service composition process. Then, once obtained the *web* services associated with the ontological concepts for every agent, the filtering process is run as a second part. The goal is to extract those services fitting the format given in the description of each agent, that is: by checking *input*, *output* and the process each one carries out. Particularly, to check a *web* service meets the ontological requirements, its file WSDL is also analyzed to verify the list of input and output semantic concepts, preconditions and actions. The *web* services filtered by the *search system* have been shown in Table A8 (Section A.2 in Appendix A). With the above filtered process

is completed the process of analysis and discovery invoked by functionality *analyze&Discover,* which returns diagram BPMN and the list of *web* services associated with each activity.

6.4. Validation of the solution

When functionality *analyze&Discover* completes its task, the tool has found the *web* services associated with the modules defined by the user and has built diagram BPMN. Then, to build the solution *web* service it is necessary:

1. For every activity, selecting the *web* service to use in diagram BPMN (note that in Table A8, Section A.2 in Appendix A, there is more than one *web* service for a module).
2. Connecting inputs and outputs for each activity in diagram BPMN.

Hence, the user should select a *web* service from list associated with each activity in the diagram BPMN by using the graphic assistant of the tool. That is, the graphic assistant of the diagram BPMN has three sections representing the information to process by the tool (see example in Figure 14):

1. *Diagram BPMN:* This section displays diagram BPMN from the XML file returned by *analyze&Discover*. The tool graphic interface allows us selecting and moving the components present in the diagram.
2. *List of activities:* This section displays the list of activities integrating diagram BPMN (associated with the modules defined by the user). For every activity has been included the following information:
    a. Activity name
    b. Performed action
    c. Domain
    d. Input list
    e. Output list
    f. List of preconditions to meet for the activity performs well.

3. *Found web services:* List of *web* services available for the selected activity. By default, the first *web* service in the list has been selected by tool, but the user can select any other from the list.

**** Figure 14 here ****

The *web* services selected for each activity/module of our case study have been shown in Table A9 (Section A.2 in Appendix A). To complete the validation process of the solution proposed by the tool and build the expected *web* service, the assistant of the diagram BPMN creates the data flow between the activities of such a diagram. That is, each activity in the diagram receives certain inputs that can come from the before activity or the input of the *web* service to build.

6.5. Composition of the solution web service

Completed the association process of *web* services with the activities of diagram BPMN and stated their connections; the tool can carry out the *web* service composition by calling functionality *composeService* (see Section 4.1). This functionality applies standard WS-BPEL and the algorithm given in [56] to make the composition process. Basically, *composeService* invokes the *manager* agent by passing diagram BPMN. The *manager* agent processes the information and calls the *composition coordinator* agent of the *composition system*. The above agent builds an agent for each *web* service (or activity) of diagram BPMN. These new agents are responsible for achieving the necessary data from files WSDL (*web* service description) for the composition process (see Figure 12). Finally, the *composition coordinator* agent receives all the information from files WSDL to generate file WS-BPEL. This file has been generated by modeling the components in composition BPEL for the solution *web* service, namely:

1. Flow control;
2. Beginning of the service;

3. Exchange of massages and
4. Events.

6.6. Evaluation of the case study

In this subsection we evaluate tool *LibraryBookReserve* and the solution *web* service generated from the composition process. The evaluation has been made taking into account concepts of *software engineering* to check the internal quality of the composition process. The internal quality of a computer program is related to *white-box testing* [57], which is the examination of procedural details of the software. It evaluates logical paths in the software by providing specific sets of conditions and/or loops. Moreover, the state of the software on different points of its implementation can be verified in order to determine whether the current state of the software matches the expected state.

For our case study, we have used *basis path testing* as a *white-box testing*. This test has been introduced to evaluate the different logical paths of the flow diagram used to build the solution *web* service through tool *LibraryBookReserve*. Therefore, it measures the performance of the internal logic structure of the solution *web* service that has automatically been built by the tool. With this test we want to prove that the automatic process of *web* service composition performs well.

6.6.1. Basis path testing for the solution web service (white-box testing)

Basis path testing is a technique of *white-box testing* proposed in [58], which allows test case designers to create a measure evaluating the logic complexity of a procedural design and uses it as a guide in the definition of a basis set of running paths. The goal of this test is to determine the number of independent paths for a set of statements (in our case, the flow diagram of the solution *web* service) to create test cases ensuring the running of each logical path, at least once. This will prove correctness and completeness of the reached solution.

To evaluate the flow diagram of the solution *web* service as shown in Figure 15, it has been created a test case for every possible path of such a diagram. This figure displays the flow diagram connecting the *web* services that integrate the service composition (see Tables

A5, A6 and A9; Section A.2 in Appendix A) to obtain the solution specified by the user. Note that this flow diagram represents the tasks created by the tool to run the solution *web* service, being these tasks, the *web* services integrating the service composition. Then, from this diagram, all possible logical paths have been computed by building the associated flow graph given in Figure 16.

**** Figure 15 here ****

Graph *G* in Figure 16 shows the flow graph associated with the flow diagram in Figure 15. Thus, this graph represents a logic control structure more suitable than the flow diagram to compute all logical paths [57]. In general terms, nodes in the graph mean one or more procedural statements and numbers identify such statements in the flow diagram. Contiguous statements can be in the same node. Arcs represent the control flow and mean the same as in the flow diagram. Predicate nodes represent conditional statements from the flow diagram and can be identify because two or more arcs emerge from them as shown in Figure 15. In this case, the predicate nodes have been identified in the graph with numbers 2, 7 and 8. Then, from this graph we can compute *cyclomatic complexity* in order to determine the number of running independent paths of the flow diagram. Cyclomatic complexity is a software metrics providing an upper bound for the number of tests that must be conducted to ensure that all statements have been executed at least once. An independent path is any path in the graph (or flow diagram) that introduces, at least, a new set of statements.

**** Figure 16 here ****

In consequence with the above, cyclomatic complexity *V* of graph *G* in Figure 16 can be computed according to [57], from expression $V(G) = A - N + 2$, where *A* is the number of arcs in the graph and *N* is the number of nodes. Then, $V(G) = 11 \text{ arcs} - 8 \text{ nodes} + 2 = 5$. Hence, we can now determine the basis set of linearly independent paths (which has 5

paths). Table 3 lists the five linearly independent paths from the flow graph, for which we have prepared a test case for each.

*** *Table 3 here* ***

For each path in Table 3, a test case has been prepared in order to force the solution *web* service to run it. As a result, the test cases have proven that each path has successfully been run at least once and the results reached in each case match the expected ones. Therefore, the logic implanted by the flow diagram in Figure 15 (which is based on the user requirements) and the selected *web* services have performed well. Additionally, all the previous process of *white-box testing* has also been applied to each flow diagram different from the one given in Figure 15, for which the same results have been obtained. Note that in the process of *web* service composition (diagram BPMN); the tool allows us to choose a *web* service different from the current one, which generates a new flow diagram to check.

6.7. Results

In this section we have introduced a case study to present an implementation of our proposal, architecture IPCASCI, which states a model to automatically compose business process. As a first result of the case study, we have given an implementation of IPCASCI as tool *LibraryBookReserve,* which creates new *web* services from the composition of *web* services existing in the platform to reserve books in a library. This result proves the reliability of our proposal, architecture IPCASCI, on a real environment of *cloud computing*. On the other hand, we have also introduced a validation process of the case study by evaluating the internal logical structure that the solution *web* service has to run (*white-box testing*). Since in our case, *white-box testing* focuses on the control structure of the solution *web* service; we have also proven the completeness and correctness of tasks performed by the flow diagram run from the process of *web* service composition (diagram BPMN, Figure 15) to obtain the expected solution. In this sense, the solution *web* service generated by the tool fits the requirements introduced by the user. Finally, all these results also prove the

goodness and reliability of the logic structure of the multi-agent system coupled to IPCASCI, which is aimed at the discovery and refinement of *web* services taking place in the services composition process.

7. Conclusions

This paper has proposed an architecture called IPCASCI for agile and automatic composition of business processes on a *cloud computing* environment. This proposal is able to fit the technological environment, being accessible and useful for the business world. For this approach, the software components to reuse in the composition process have been *web* services. In this sense, the main characteristic of our proposal is that from a set of specifications (in text format) given by an inexperienced user to build a business process; IPCASCI is able to develop the product as a *web* service from the ones existing in the platform. To do this, IPCASCI has provided a *web* service repository in a *cloud computing* environment; an ontology system in order to represent the necessary knowledge to endow semantics (WSDL-S, Web Service Semantics Language) to the existing *web* services, and so later facilitate their discovery process and composition; a multi-agent system coupling agent-based virtual organizations aimed at discovering and composing the *web* services fitting the requirements initially given by the user.

To prove the performance of this proposal, we have presented an implementation of IPCASCI as a tool from a real case study. Therefore, our first result of the case study has been to develop tool *LibraryBookReserve* to automatically build *web* services from specifications of the user to reserve books in a library. According to the requirements of the user, this tool provides a graphic interface to introduce such specifications in a simple, intuitive and agile way. In this context, we have also proven through a *white-box testing* (software metrics), that the internal logic structure of the solution *web* service (in this case, its flow diagram) built by *LibraryBookReserve* performs well on the selected application domain. Moreover, by simple modifications of the tool, we can add other application domains, extending its use and so providing reusability. Consequently, this proves that

architecture IPCASCI is not only useful for the presented case study; it is also useful for any other application domain. Hence, our proposal adapts to the current needs of software development, for which it is possible to create new products in very short time, at a reduced cost and without going through an arduous and external development process.

Finally, keep in mind that our approach has been benefited from the potential of a *cloud* environment, which introduces a change in the way of exploiting and marketing the company's products [59]. Hence, IPCASCI deals with the following advantages present in systems based on *cloud computing*: (i) Resources in a *cloud* environment can be allocated and dissallocated according to needs at any given time; (ii) Resources can be shared or/and allocated to each customer depending on their needs; (iii) Services stored in the cloud are generally based on *web* technology, a fact that makes them accessible to a broad range of devices with Internet connection; (iv) Once *cloud* service infrastructure is outsourced, the service provider transfers its business risks (as for example, hardware failures) to the infrastructure providers that are usually more experienced and better equipped to handle such risks.

Author contributions

JAGC designed and implemented the proposed architecture supervised by JMC. JAGC and JAC-G wrote the paper. JMC and JAC-G provided the comments and the discussion. All authors read and approved the final manuscript.

Competing interests

The authors declare that they have no competing interests.

Appendix A. Information supporting architecture IPCASCI

A.1. Example of converting the agent subsystem into a BPD diagram

This example shows the transformation of the modules and their relationships defined by the user into an *agent subsystem* and later into the BPD diagram. Then, in this case we have defined seven modules, from Module1 to Module7, where modules 1 and 7 are the ones of startup and end respectively. The following relationships between the introduced modules have been defined:

1. *Modulo1 Parallel output to Modulo2, Modulo3*
2. *Modulo2 If [Condition] Modulo4 Else Modulo5*
3. *Modulo6 Follows Modulo4*
4. *Modulo6 Follows Modulo5*
5. *Modulo7 Follows Modulo6*
6. *Modulo7 Follows Modulo3*

The *agent subsystem* created by the *analysis system* corresponding to the above input has been shown in Figure A1. This *agent subsystem* has been mapped to standard BPMN in Figure A2 by applying the transformations given in Table 1, Section 5.1.

*** Figure A1 here ***

*** Figure A2 here ***

A.2. Complementary information of the case study

*** Table A1 here ***

*** Table A2 here ***

*** Table A3 here ***

*** Table A4 here ***

*** *Table A5 here* ***

*** *Table A6 here* ***

*** *Table A7 here* ***

*** *Table A8 here* ***

*** *Table A9 here* ***

*** *Figure A3 here* ***

*** *Figure A4 here* ***

References

[1] K. Schmid, Top Productivity through Software Reuse, 12th International Conference on Software Reuse, Lecture Notes in Computer Science, ICSR, Springer 6727, 2011.

[2] J. S. Poulin, Measuring software reuse. Principles, practices, and economic models, Addison-Wesley-Longman, ISBN-978-0-201-63413-6, 1997.

[3] J. S. Poulin, The Business Case for Software Reuse: Reuse Metrics, Economic Models, Organizational Issues, and Case Studies, Lecture Notes in Computer Science, Springer 4039 (2006) 439.

[4] M. Lemley and D. O'Brien, Encouraging Software Reuse, Stanford Law Review 49 (1997) 255.

[5] R. D. Shang, K. Mohan, K. R. Lang, R. Vragov, A market mechanism for software component reuse: opportunities and barriers, Proceedings of the 14th Annual International Conference on Electronic Commerce (ICEC '12), ACM (2012) 62-69.

[6] H. Rehesaar, Capability Assessment for Introducing Component Reuse, Lecture Notes in Computer Science, Springer-Verlag 6727 (2011) 87-101.

[7] Y. Xu, N. Singh, S. Deshpande, Reuse by Placement: A Paradigm for Cross-Domain Software Reuse with High Level of Granularity, Lecture Notes in Computer Science, Springer 6727 (2011) 69-77.

[8] V. C. Garcia, D. Lucrédio, A. Alvaro, E. S. de Almeida, R. P. de Mattos, S. R. de Lemos, Towards a maturity model for a reuse incremental adoption, In: The 1st Brazilian Symposium on Software Components, Architecture and Reuse, Campinas, Sâo Paulo, Brazil (2007) 61-74.

[9] L. Osiecki, M. Phillips, J. Scibilia, Understanding and Leveraging a Supplier's CMMI Efforts: A Guidebook for Acquirers, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, Technical Report CMU/SEI-2011-TR-023, 2011

[10] A.E. Walsh, UDDI, SOAP, and WSDL: The Web Services Specification, Pearson Education, 1st edition, 2002.

[11] T. Erl, SOA Design Patterns, The Prentice Hall Service-Oriented Computing Series, 2009.

[12] J. Ralyté, I. Mirbel, R. Deneckère, Engineering Methods in the Service-Oriented Context, 4th IFIP WG 8.1 Working conference on method engineering, 2011.

[13] K. Alizadeh, M. Seyyedi, M. Mohsenzadeh, A Service Identification Method based on Enterprise Ontology in Service Oriented Architecture, International Journal of Information Processing and Management (IJIPM) 3 (2012).

[14] M. Papazoglou, Web Services: Principles and Technology, Pearson Education, Prentice Hall, 2008.

[15] N. Antonopoulos, A. Anjum and L. Gillam, Intelligent techniques and architectures for autonomic clouds, Journal of Cloud Computing: Advances, Systems and Applications (2012) 1-12.

[16] A. Stage and T. Setzer, Network-aware migration control and scheduling of differentiated virtual machine workloads, In: Proceedings of the ICSE Workshop on Software Engineering Challenges of Cloud Computing (CLOUD '09), IEEE Computer Society, Washington, DC, USA (2009) 9-14.

[17] T. Duy, Y. Sato, Y. Inoguchi, A prediction-based green scheduler for datacenters in clouds, IEICE Trans Inf Syst E94-D 9 (2011) 1731-1741.

[18] Cloud Computing, Retrieved at June 03, [http:/ / fclose.com/ b/ cloud-computing/ article/ mrcc-a-distributed-c-compiler-syste m-on-mapreduce/], 2011.

[19] EuroCloud Deutschland_eco eV, Eurocloud star audit saas certificate, http://www.saas-audit.de, 2011.

[20] P. T. Jaeger, J. Lin, J. M.Grimes, S. N. Simmons, Where is cloud? Geography, Economics, Environment, and Jurisdiction in Cloud Computing. ISSN-1396–0466, 4 (2009).

[21] F. Dölitzscher, C. Reich, A. Sulistio, Designing Cloud Services Adhering to Government Privacy Laws, In: Proceedings of 10th IEEE International Conference on Computer and Information Technology (CIT), Furtwangen, Germany (2010) 930-935.

[22] P. Massonet, S. Naqvi, C. Ponsard, J. Latanicki, B. Rochwerger, M. Villari, A Monitoring and Audit Logging Architecture for Data Location Compliance in Federated Cloud Infrastructures, In Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), IEEE International Symposium on (2011) 1510-1517.

[23] M. Schaaf, A. Koschel, S .G. Grivas, I. Astrova, An active DBMS style activity service for cloud environments. Cloud Computing: The First International Conference on Cloud Computing, GRIDs, and Virtualization in Cloud Computing, ComputationWorld IARIA, ISBN: 978-1-61208-106-9 (2010) 80-85.

[24] A. Sulistio, C. Reich and F. Dölitzscher, Cloud Infrastructure & Applications – CloudIA, In: Proceedings of the 1st International Conference on Cloud Computing (CloudCom), Beijing, China (2009) 583-588.

[25] S. Rodríguez, Y. de Paz, J. Bajo and J. M. Corchado, Social-based Planning Model for Multiagent Systems, Expert Systems with Applications 38 (2011) 13005-13023.

[26] V. Dignum, J. Vazquez-Salceda, F. Dignum, OMNI: Introducing Social Structure, Norms and Ontologies into Agent Organizations. Proceeding of 3rd International Workshop on Programming Multi-Agent Systems, Utretch, The Netherlands (2005) 181-198.

[27] M. Escriva, J. Palanca, G. Aranda, A. García, V. Julian, V. Botti, A Jabber-Based Multi-Agent System Platform, Proceeding of 5th International Joint Conference on Autonomous Agents and Multiagent Systems, Hakodate, Japan (2006) 1282-1284.

[28] A. Maedche and S. Staab, Learning Ontologies for the Semantic Web, In Semantic Web Workshop, Hongkong, China, 2001.

[29] B. Chandrasekaran and J. R. Josephson, What Are Ontologies, and Why Do We Need Them?, Ohio State University V. Richard Benjamins, University of Amsterdam, 1999.

[30] N. Noy and D. McGuinness, Ontology development 101: A guide to creating your first ontology, Technical Report SMI-2001-0880, Stanford Medical Informatics (SMI), Department of Medicine, Stanford University School of Medicine, 2001.

[31] N. Guarino, Formal Ontology in Information Systems, Proceedings of FOIS'98, Trento, Italy, 6-8 June, Amsterdam, IOS Press, 1998.

[32] F. López, Overview of Methodologies for Building Ontologies, In Proceedings of IJCAI-99 workshop on Ontologies and Problem-Solving Methods (KRR5) in Stockholm Sweden, August 2, 1999.

[33] C. Pedrinaci, C. Brelage, T. Van Lessen, J. Domingue, D. Karastoyanova, F. Leymann, Semantic business process management: scaling up the management of business processes, In: 2nd IEEE International Conference on Semantic Computing (ICSC), 4-7 Aug 2008, Santa Clara, CA, USA, 2008.

[34] Z. Su, H. Chen, L. Zhu, Y. Zeng, Framework of Semantic Web Service Discovery Based on Fuzzy Logic and Multi-phase Matching, Journal of Information & Computational Science 9:1 (2012) 203-214.

[35] K. Sycara, M. Paolucci, A. Ankolekar, N. Srinivasan, Automated discovery, interaction and composition of Semantic Web services. Web Semantics: Science, Services and Agents on the World Wide Web, Elsevier 1 (2003) 27-46.

[36] J. M. García, D. Ruiz, A. Ruiz-Cortés, Improving Semantic Web Services Discovery Using SPARQL-Based Repository Filtering, Web Semantics: Science, Services and Agents on the World Wide Web, Elsevier 17 (2012) 12-24.

[37] M. Owen and J. Raj, BPMN and Business Process Management. Introduction to the New Business Process Modeling Standard, Popkin Software, www.popkin.com, 2003.

[38] J. Pasley, How BPEL and SOA Are Changing Web Services Development, Published by the IEEE Computer Society, IEEE Internet Computing (2005) 60-67.

[39] M. Schaaf, A. Koschel, S. G. Grivas, I. Astrova, An active DBMS style activity service for cloud environments. Cloud Computing: The First International Conference on Cloud Computing, GRIDs, and Virtualization in Cloud Computing, in ComputationWorld IARIA (2010) 80-85.

[40] S. Rodríguez, D. Tapia, E. Sanz, C. Zato, F. de la Prieta, O. Gil, Cloud Computing Integrated into Service-Oriented Multi-Agent Architecture, Balanced Automation Systems for Future Manufacturing Networks, IFIP Advances in Information and Communication Technology, Springer 322 (2010) 251-259.

[41] B. Cao, B. Li, Q. Xia, A Service-Oriented Qos-Assured and Multi-Agent Cloud Computing Architecture, Cloud Computing, Lecture Notes in Computer Science, Springer-Verlag Berlin Heidelberg 5931 (2009) 644-649.

[42] J. Bajo, C. Zato, F. de la Prieta, A. de Luis, D. Tapia, Cloud Computing in Bioinformatics, Distrib. Computing & Artif. Intell., AISC, Springer-Verlag Berlin Heidelberg 79 (2010) 147-155.

[43] IBM, RESTful Web services: The basics, http://www.ibm.com/developerworks/webservices/library/ws-restful/, 2008.

[44] E. Newcomer. Understanding Web Services: XML, WSDL, SOAP, and UDDI (independent Technology Guides). Addison Wesley, 2002.

[45] E. Cerami. Web services Essentials – Distributed Applications with XML-RPC, SOAP, UDDI & WSDL. O'RELLY, 2002.

[46] F. M. T. Brazier, V. Dignum, M. N. Huhns, C. Derksen, F. Dignum, T Lessner, J. A. Padget, T. B. Quillinan, M. P. Singh, Agent-based organisational governance of services, Multiagent and Grid Systems 8 (2012) 3-18

[47] D. Tapia, S. Rodríguez, J. Bajo and J. M. Corchado. FUSION@, A SOA-Based Multi-agent Architecture. International Symposium on Distributed Computing and Artificial Intelligence (DCAI), Springer-Verlag Berlin Heidelberg 50 (2009) 99-107.

[48] J. Hendler, T. Berners-Lee, E. Miller, Integrating Applications on the Semantic Web, Journal of the Institute of Electrical Engineers of Japan 10 (2002) 676-680.

[49] M. d'Aquin and N. Noy, To publish and find ontologies? A survey of ontology libraries, Web Semantics: Science, Services and Agents on the World Wide Web, Elsevier 11 (2012) 96-111.

[50] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, K. Sycara, OWL-S: Semantic Markup for Web Services, W3C Member Submission, http://www.w3.org/Submission/OWL-S, 2004.

[51] J. Miller, K. Verma, P. Rajasekaran, A. Sheth, R. Aggarwal, K. Sivashanmugam, WSDL-S: Adding Semantics to WSDL, LSDIS Lab, University of Georgia, http://lsdis.cs.uga.edu/projects/meteor-s/, 2004.

[52] R. Akkiraju, J. Farrell, J. Miller, M. Nagarajan, M. Schmidt, A. Sheth, K. Verma, Web Service Semantics - WSDL-S, W3C Member Submission, http://www.w3.org/Submission/WSDL-S/, 2005.

[52] A. Arkin, S. Askary, B. Bloch, F. Curbera, Y. Goland, N. Kartha, K. Liu, S. Thatte, P. Yendluri and A. Yiu, Web Services Business Process Execution Language, Version 2.0 OASIS, 2004.

[53] C. Ouyang, W. van der Aalst, M. Dumas, A. Hofstede, Translating BPMN to BPEL, Digital Repository, Queensland University of Technology, Australia, 2006.

[54] R. Jan and M. Jan, On the Translation between BPMN and BPEL: Conceptual Mismatch between Process Modeling Languages, In Latour, Thibaud & Petit, Michael (Eds.), The 18th International Conference on Advanced Information Systems Engineering, Proceedings of Workshops and Doctoral Consortium, Namur University, 2006

[55] C. Ouvans, M. Dumas, A. Hofstede, W. van der Aalst, From BPMN Process Models to BPEL Web Services, ICWS '06 Proceedings of the IEEE International Conference on Web Services, IEEE Computer Society Washington, DC, USA (2006) 285-292.

[56] S. White, Using BPMN to Model a BPEL Process, BPTrends, http://www.bptrends.com, 2005.

[57] R. Pressman, Software Engineering: A Practitioner's Approach, Seventh Edition, Ph.D., McGraw-Hill Companies, 2005.

[58] T. McCabe, A Software Complexity Measure, IEEE Trans. Software Engineering, SE-2 (1976) 308-320.

[59] V. Chang, D. Bacigalupo, G. Wills, D. De-Roure, A Categorisation of Cloud Computing Business Models, 10th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (2010) 509-512.

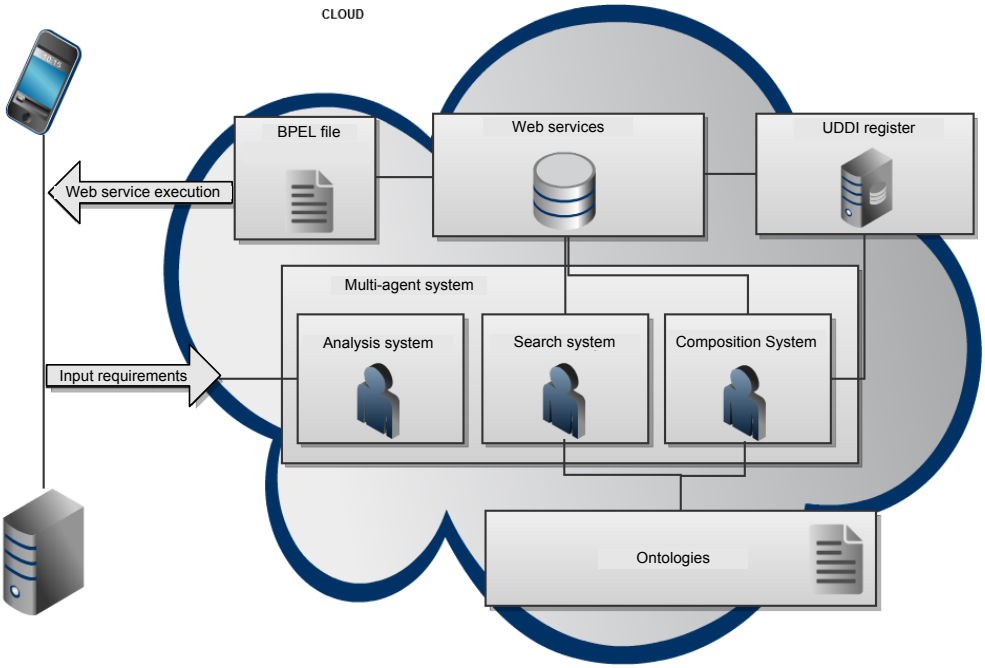

Figure 1: Description of architecture IPCASCI for composition of bussiness processes (*web* services).

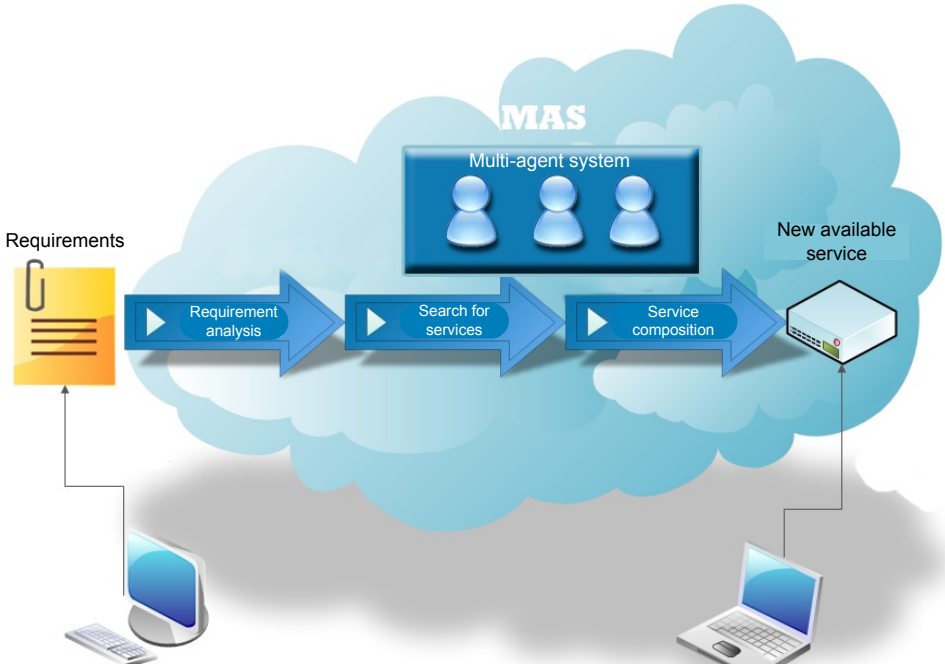

Figure 2: Stages of architecture IPCASCI to obtain the solution *web* service from the discovery and composition process of *web* services.

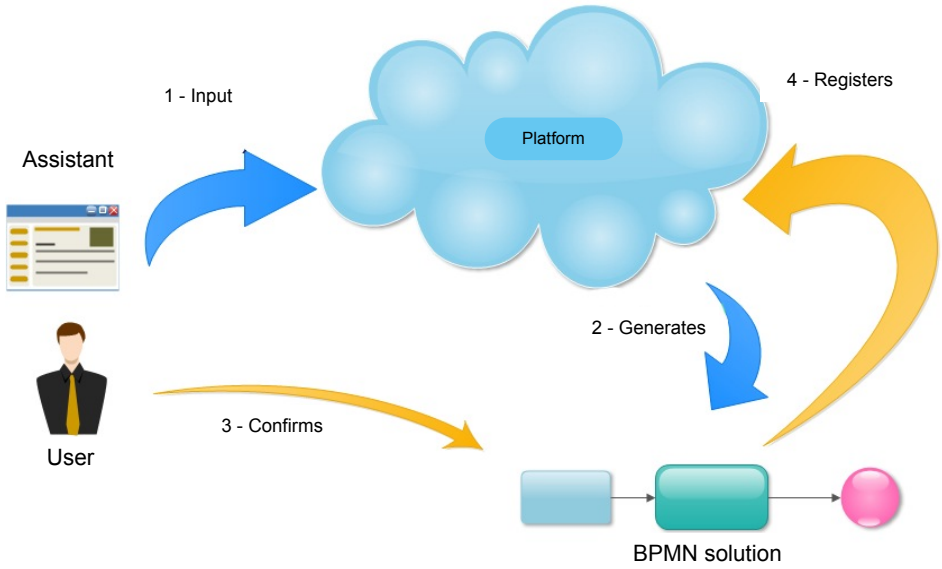

Figure 3: Alternative view (Figure 2) of the process of obtaining the solution *web* service by architecture IPCASCI.

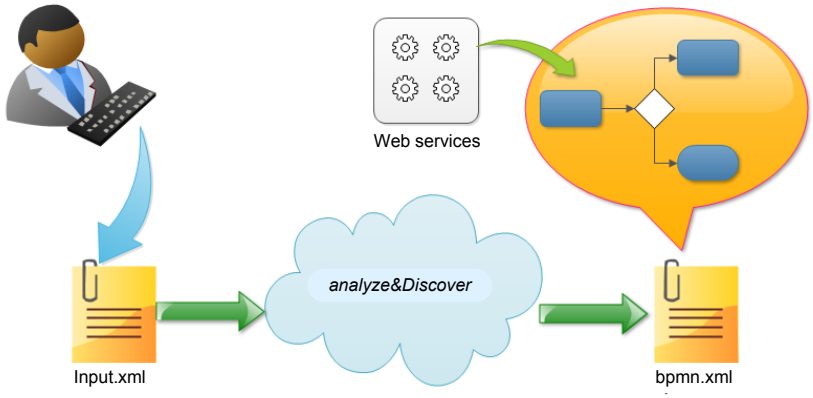

Figure 4: *analyze&Discover cloud* functionality interacting with components of architecture IPCASCI.

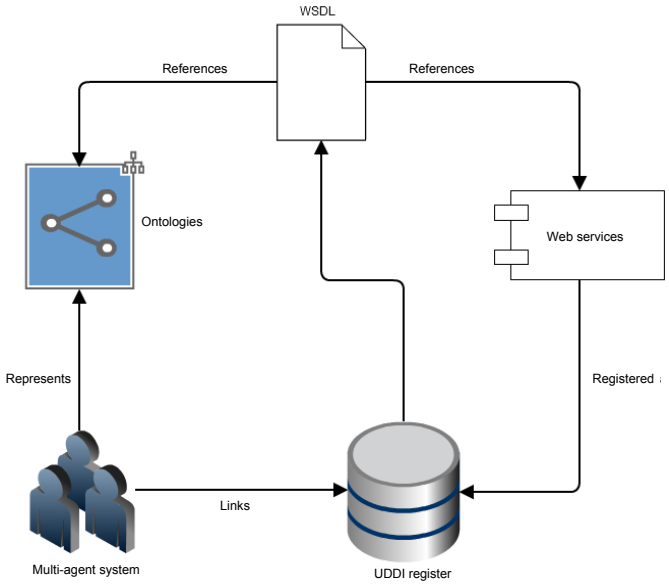

Figure 5: Integration diagram of *web* services and SOA.

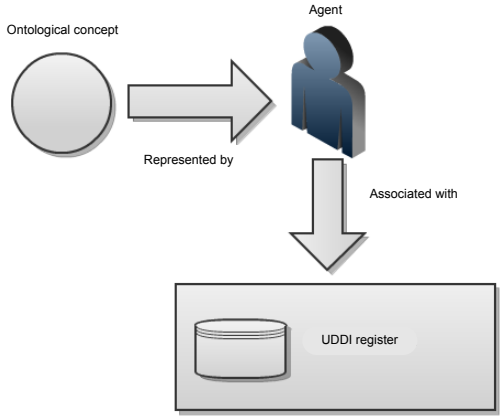

Figure 6: Connecting semantic concepts with entries in register UDDI (*web* services) through agents.

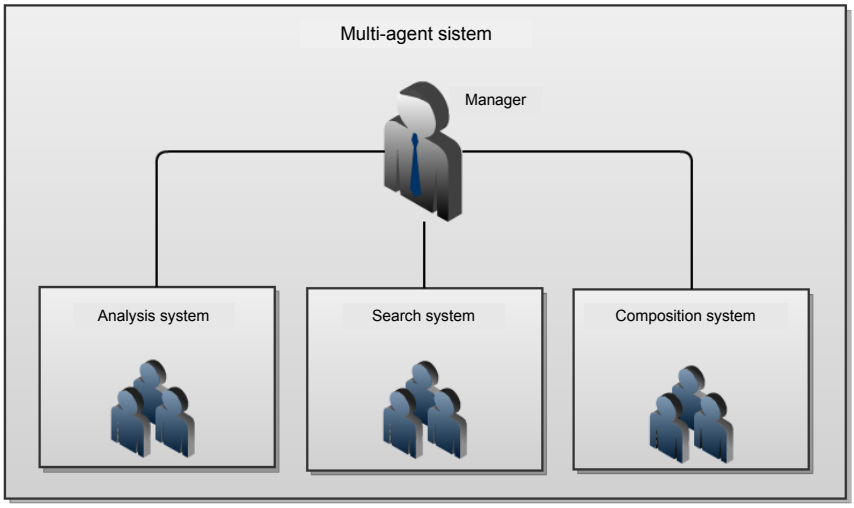

Figure 7: Structure of the multi-agent system (MAS) to represent the domain ontology and manage the requirement input, search and composition of *web* services.

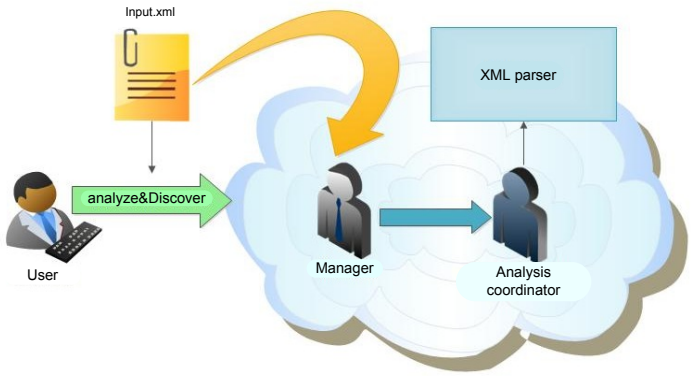

Figure 8: Agent-based virtual organization of the *analysis system* to analyze and discover the *web* services associated with the requirements of the user.

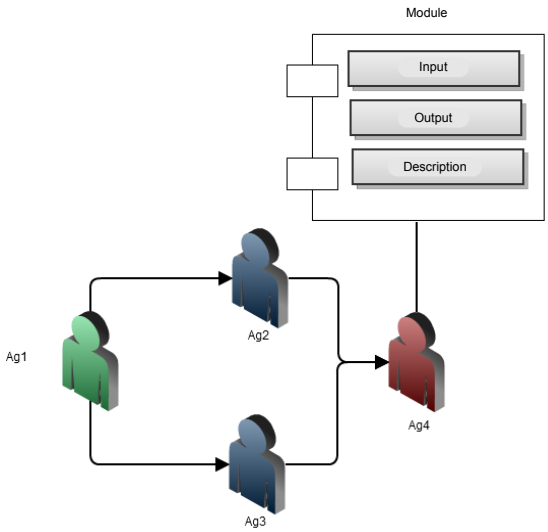

Figure 9: Relationship module-agent and the agent-based graph associated with associated with the modules defined by the user.

Table 1: Relationships between items in the BPMN diagram and items in the *agent subsystem*.

BPMN diagram	Agent subsystem
Activity	Agent
XOR gate (“X”)	*[Module_i] If [Condition] then [Module_j]* or *[Module_i] If [Condition] then [Module_j] else [Module_k]*
AND gate (“+”)	*[Module_i] Parallel output to [Module_j], [Module_k]*
startup and end events	agents marked as startup and end process
any other connector	*[Module_i] Follows [Module_j]*

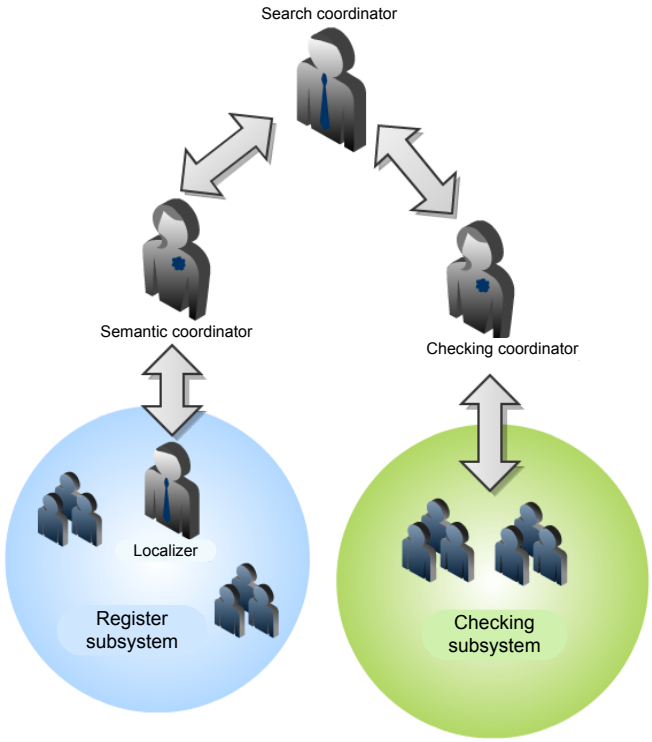

Figure 10: Agent-based virtual organization representing the structure of the s*earch system*.

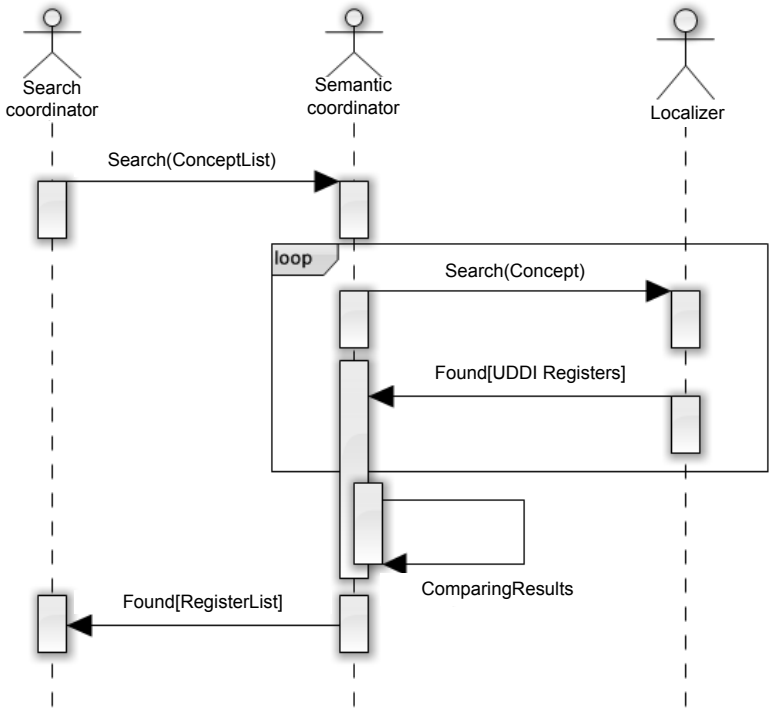

Figure 11: Process of semantics search for the *web* services associated with a list of ontological concepts.

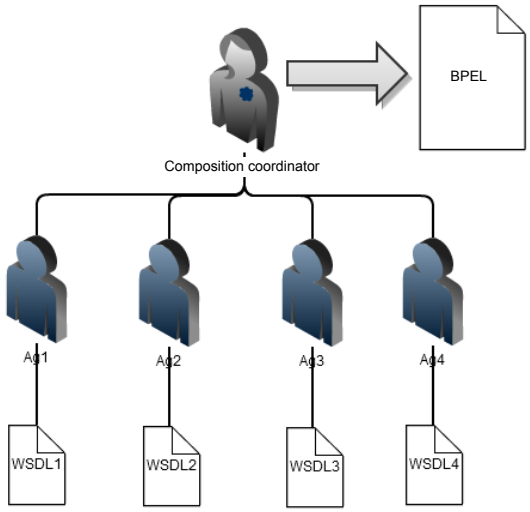

Figure 12: Agent-based *composition system* to obtain the solution *web* service.

Table 2: Definition and meaning of the modules to build the solution *web* service.

Module	Description
CheckingRecord	Determine whether the user is registered in the system.
NotifyState	Notify to the user that the connection has not been established and the reason.
CheckingBook	Verify the existence of the requested book.
ChekingLoan	Verify the maximum number of loans stated for the user.
NotifynoStock	Notify to the user that there is no existence of the requested book.
NotifyLoanExceed	Notify to the user that the maximum number of loans has been exceeded.
MakeLoan	Carry out and register the book loan.
CheckedRecord	Auxiliary module useful for the later construction of diagram BPD of graphic standard BPMN. This module is defined by the system.

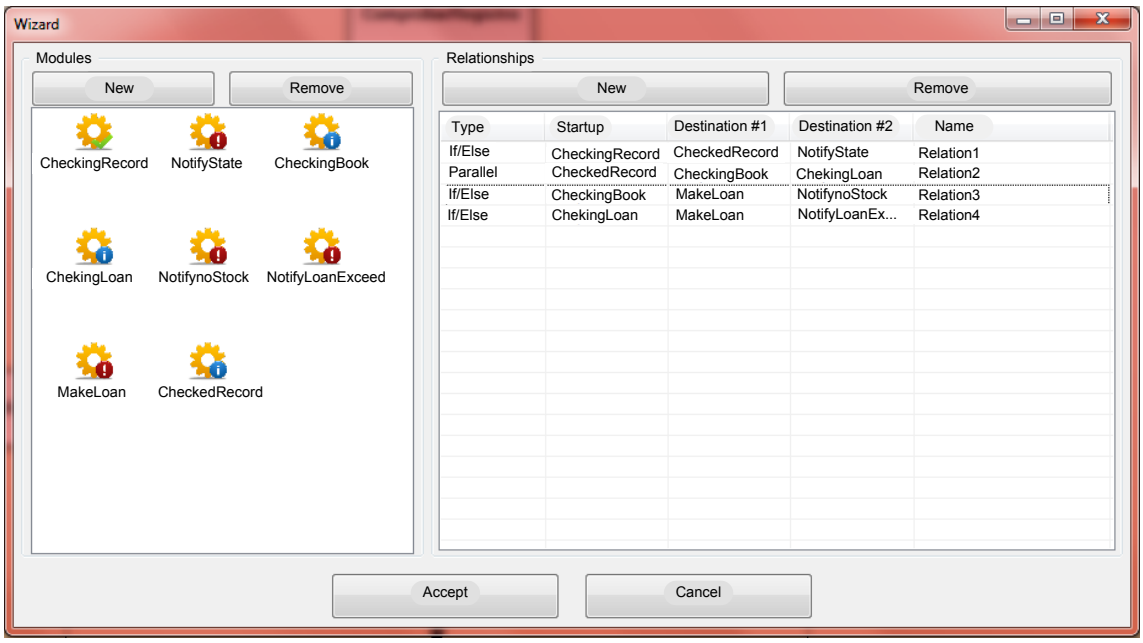

Figure 13: View of tool *LibraryBookReserve* showing on the left side, the modules defined by the user and on the right, the relationships defined between them.

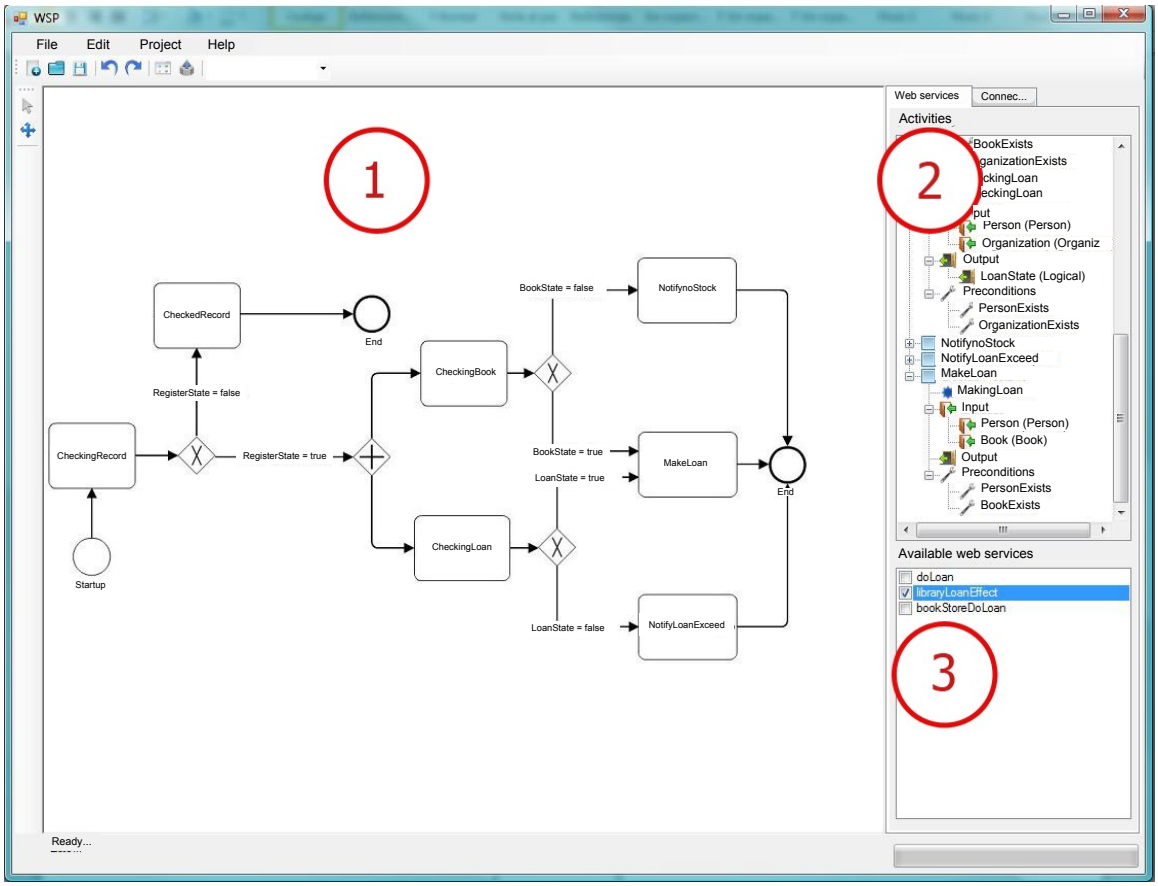

Figure 14: View of tool *LibraryBookReserve* showing an example where on Part 1 is diagram BPMN of the case study, on Part 2, the list of activities and on Part 3, the available *web* services.

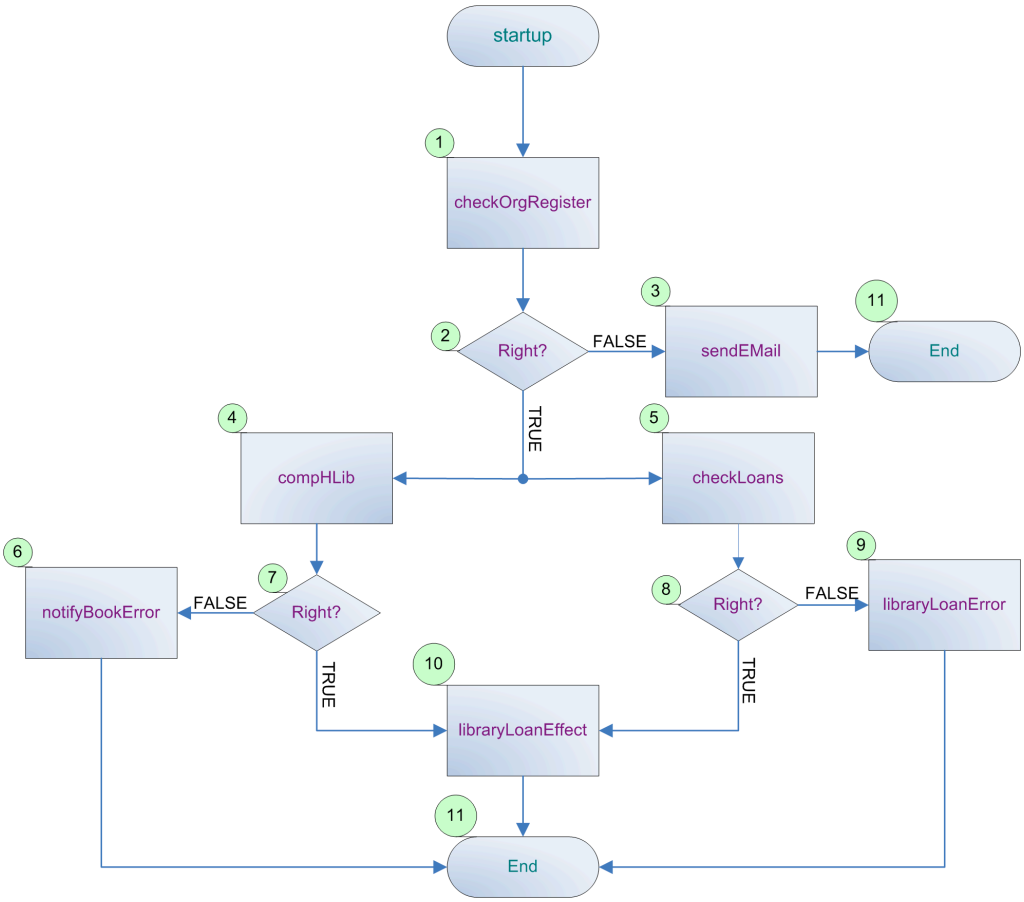

Figure 15: Flow diagram of the *web* service composition to obtain the solution *web* service. Rectangles represent the *web* services selected for the composition.

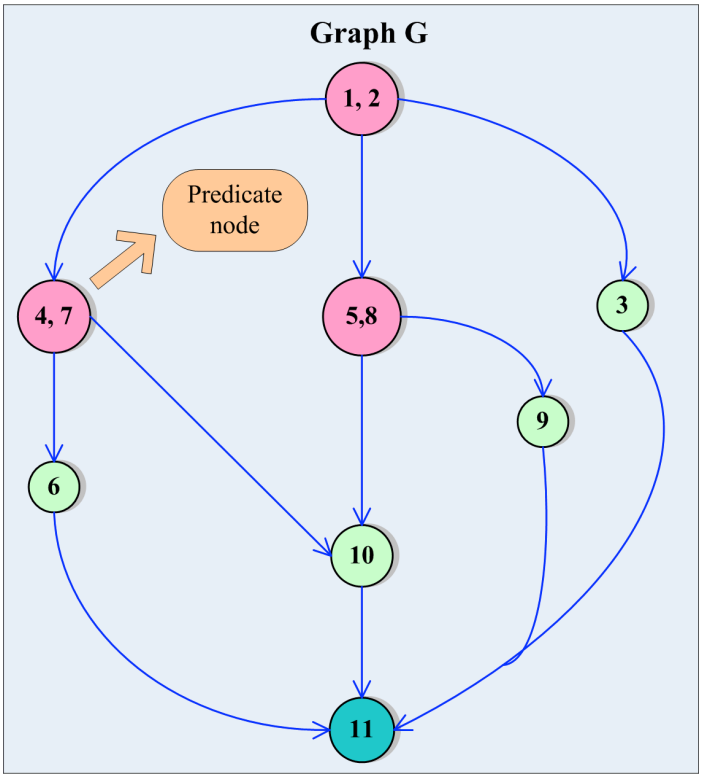

Figure 16: Flow graph associated with the flow diagram in Figure 15. Nodes and numbers represent tasks in the flow diagram.

Table 3: Linearly independent logic paths found in the flow graph in Figure 16.

Logical paths	
Path #1:	1 – 2 – 4 – 7 – 6 – 11
Path #2:	1 – 2 – 4 – 7 – 10 – 11
Path #3:	1 – 2 – 5 – 8 – 10 – 11
Path #4:	1 – 2 – 5 – 8 – 9 – 11
Path #5:	1 – 2 – 3 – 11

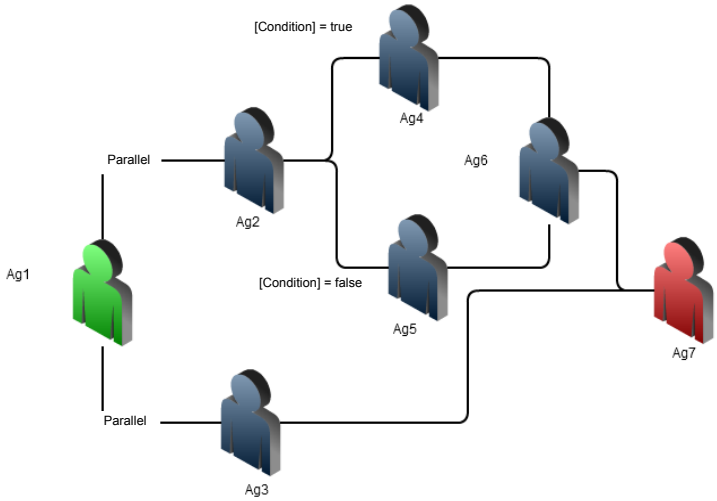

Figure A1: *Agent subsystem* representing a domain ontology built by the *analysis system*.

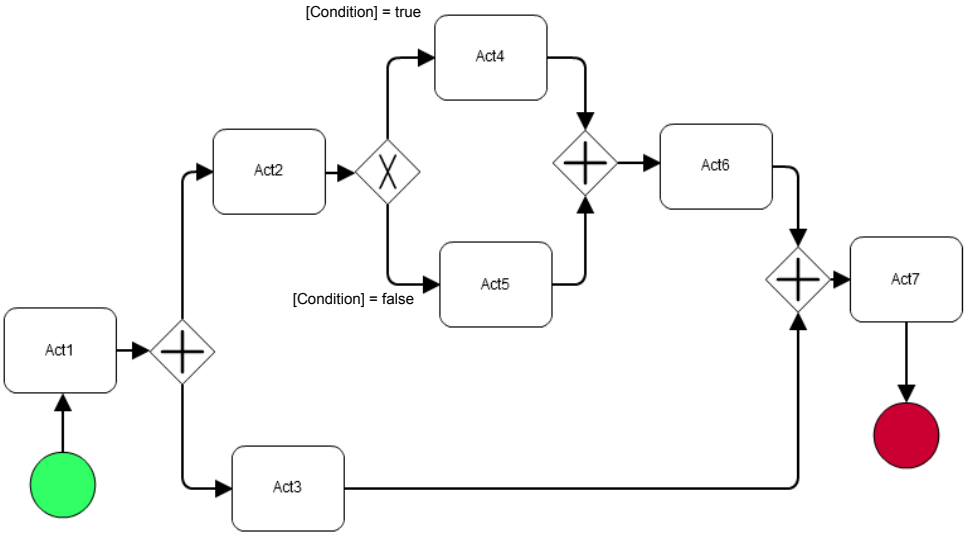

Figure A2: Diagram BPMN (BPD) mapped from the *Agent subsystem* in Figure A1.

Table A1: Semantics description of the attributes for modules CheckingRecord and NotifyState.

Name:	*CheckingRecord*	*Name:*	*NotifyState*
Input:	Person Organization	Input:	person
Output:	recordState	Output:	-

Precondition:	organizationExists	Precondition:	personExists
Action:	verifyRecord	Action:	sendEmail
Domain:	authentication	Domain:	message
Type:	startup	Type:	end

Table A2: Semantics description of the attributes for modules CheckingBook and ChekingLoan.

Name:	*CheckingBook*	*Name:*	*ChekingLoan*
Input:	book Organization	Input:	person organization
Output:	bookState	Output:	loanState
Precondition:	bookExists organizationExists	Precondition:	personExists
Action:	checkingAvailableBook	Action:	checkingLoan
Domain:	Library	Domain:	library
Type:	Intermediate	Type:	intermediate

Table A3: Semantics description of the attributes for modules NotifynoStock and NotifyLoanExceed.

Name:	*NotifynoStock*	*Name:*	*NotifyLoanExceed*
Input:	person book	Input:	person
Output:	-	Output:	-
Precondition:	bookExists personExists	Precondition:	personExists
Action:	notifyNoBooks	Action:	notifyNoLoan
Domain:	library	Domain:	library
Type:	end	Type:	end

Table A4: Semantics description of the attributes for module MakeLoan.

Name:	*MakeLoan*
Input:	person

	book
Output:	-
Precondition:	personExists
	bookExists
Action:	makeLoan
Domain:	library
Type:	End

Table A5: Semantics description of relationships Relation1 and Relation2 defined on the modules introduced by the user.

Name:	*Relation1*	*Name:*	*Relation2*
Type:	If/Else	Type:	Parallel
Startup:	CheckingRecord	Startup:	CheckedRecord
Destination1:	NotifyState	Destination1:	CheckingBook
Destination2:	CheckedRecord	Destination2:	ChekingLoan
Condition:	registerState	Condition:	-

Table A6: Semantics description of relationships Relation3 and Relation4 defined on the modules introduced by the user.

Name:	*Relation3*	*Name:*	*Relation4*
Type:	If/Else	Type:	If/Else
Startup:	CheckingBook	Startup:	ChekingLoan
Destination1:	MakeLoan	Destination1:	MakeLoan
Destination2:	NotifynoStock	Destination2:	NotifyLoanExceed
Condition:	bookState	Condition:	loanState

Table A7: *Web* services available in the platform for the case study of architecture IPCASCI.

Web services	Function
checkPersonRegister checkOrgRegister	Check whether a user is registered in the system.

SendMail sendEMail emailManager	Send and manage the electronic mail (e-mail).
libraryManager checkBook compHLib	Identify books and manage their availability in the library.
stockManager loanManager checkLoans loanController	Check and manage the (maximum) number of books loaned to a user.
sendNotAvaliableBook notifyBookError	Notify that there is not availability of a book or any other problem preventing the loan.
loanPassed notifyLoans libraryLoanError	Notify that the maximum number of book loans has been reached.
doLoan libraryLoanEffect bookStoreDoLoan	They are used to register a book loan by the library.

Table A8: *Web* services found and checked by the *search system* for each module defined by the user.

Modules	Found *web* services
CheckingRecord	checkPersonRegister checkOrgRegister
NotifyState	sendMail sendEMail
CheckingBook	compHLib
ChekingLoan	checkLoans
NotifynoStock	sendNotAvaliableBook notifyBookError
NotifyLoanExceed	notifyLoans libraryLoanError

Table A9: *Web* services selected for each defined module.

Activity/Module	Found *web* services
CheckingRecord	checkPersonRegister
NotifyState	sendEMail
CheckingBook	compHLib
ChekingLoan	checkLoans
NotifynoStock	notifyBookError
NotifyLoanExceed	libraryLoanError

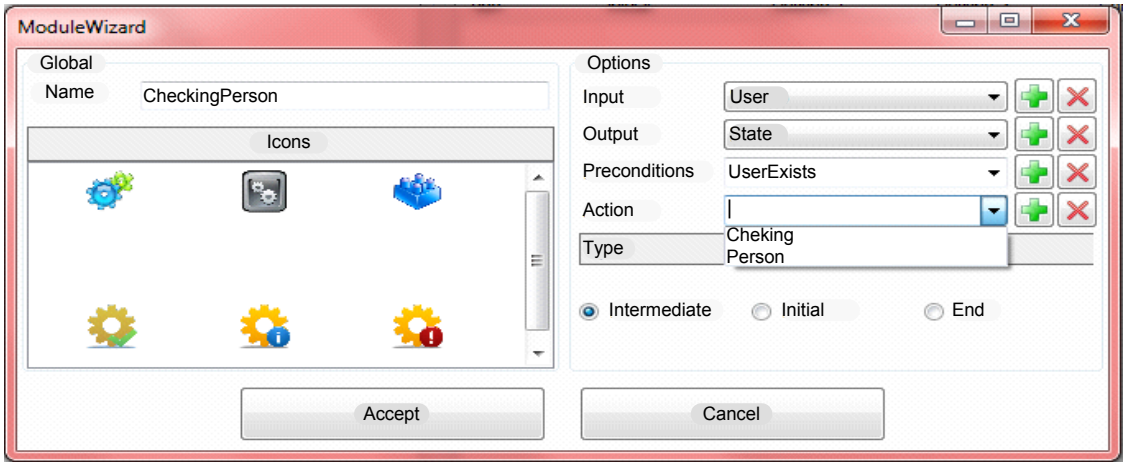

Figure A3: View of tool *LibraryBookReserve* showing an example to define a module and its semantics.

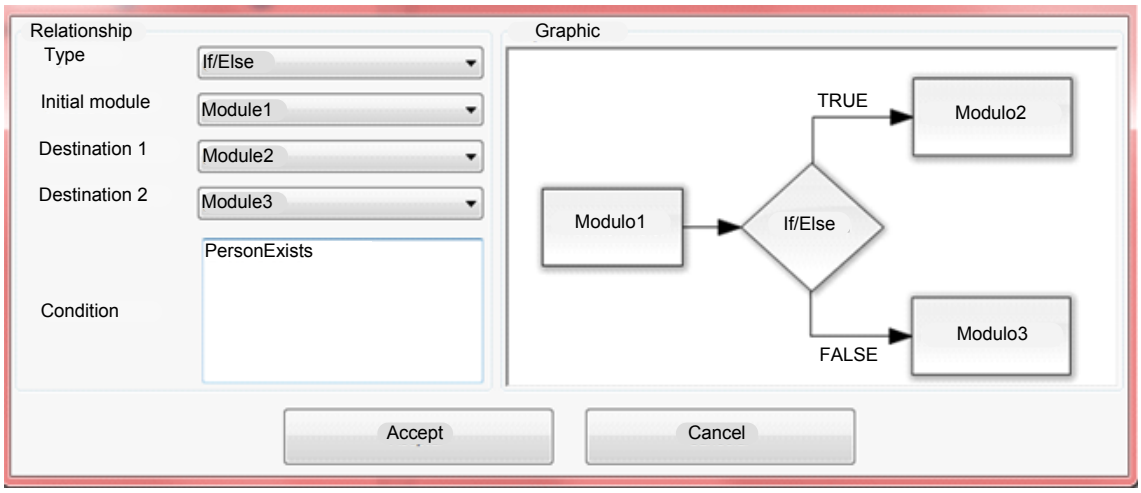

Figure A4: View of tool *LibraryBookReserve* showing an example to define the relationships between the built modules.