A Tool for Automatic Generation of WS-BPEL Compositions from OWL-S described services

Luca Bevilacqua[2], Angelo Furno[1], Vladimiro Scotto di Carlo[2], Eugenio Zimeo[1]

[1] Department of Engineering, University of Sannio, Benevento, 82100 Italy
[2] Engineering, Napoli, 80142 Italy

Service composition is a fundamental facet of Service Oriented Architecture to burst the creation of new services and knowledge throughout the Internet. Automating this aspect has been for many years an interesting research topic for people working in several research areas. In spite of the several scientific results already achieved, generating a concrete and runnable service composition from the semantic descriptions of the domain services and the problem to solve is still an open issue. This paper presents an approach to automatic service composition in the context of autonomic workflows and a related tool developed for an IT industrial context. The tool is able to retrieve service descriptions from a repository, to support the definition of the problem to solve, to generate an abstract plan and to translate it into an executable process language, such as WS-BPEL. This way, the tool covers the overall life-cycle of autonomic workflows, from their inception to the adaptive execution. The paper compares the approach with other proposals and shows its effectiveness through a case study that exploits automatic service composition to handle an emergency situation caused by a hydrogeological disaster.

***Index Terms*— SOA, Semantic Web Services, Automatic Composition, Autonomic Workflow, Business Process Generation.**

I. Introduction

The adoption of Web services and Service Oriented Architectures is promoting a novel approach for developing web applications, since they can be created by composing distributed services hosted by servers in different administration domains. We refer to this new kind of large-scale, distributed application as “multi-organization Web application”.

With the term large-scale, we intend the involvement of a large numbers of services available throughout the Internet, annotated with semantic descriptions. Services can be modified or replaced; they can disappear, and new services with different features may become available.

As multi-organization Web applications integrate a variety of heterogeneous IT systems, they introduce a new level of complexity: their design requires the knowledge of an enormous amount of details and events to be handled for ensuring a correct execution in the presence of anomalies generated by improper actions or unexpected events.

This new level of exception handling needs features that are beyond the capability of current tools for the development and management of applications. To handle this level of dynamicity, autonomic computing (AC) represents a viable solution. As it allows systems to manage themselves, service compositions can benefit from this approach to properly react to external events in order to change their structure accordingly, reducing human intervention to the minimum.

On the basis of these mechanisms, we have defined the concept of autonomic workflow [1][27], a composition of automatic or manual services that is able to proceed towards the goal even if external events significantly change the execution context. To survive to changes, a service composition needs to be modified, taking into account the new environment. In this context, an important role is performed by the *configurator*, a component of an autonomic composition engine that is in charge of implementing self-configuration of service compositions through the knowledge coded at design-time or collected at run-time. The configurator acts on every aspect of a concrete service composition by changing the overall composition graph to make it runnable within the new conditions. To this end, it can: change a link between an activity and a concrete service (re-bind); insert, delete or replace an activity; change the endpoints of a transition; substitute an activity with a sub-process that is able to perform the same actions and to produce the same effects on the external world.

The configurator exploits some internal components to generate or change compositions (see Fig. 1).

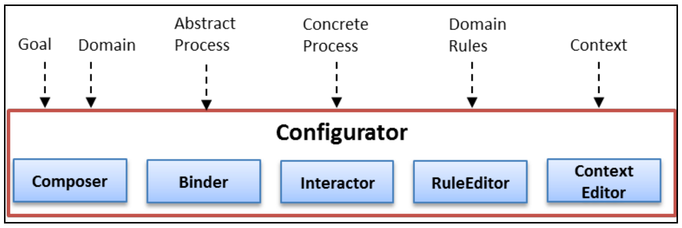

Figure 1. Configurator component of a workflow engine

One of the main components of the configurator is the *composer*. It can be used either for the initial definition (plan) of a service composition or to re-plan an already defined composition, which needs to be changed completely or in part to react to external events. The composer exploits planners to transform the descriptions of a goal and a domain in an abstract process. This can be further concretized through the binder, a component in charge of linking an abstract activity with a concrete service. Moreover, domain rules can be exploited to validate automatic compositions generated by the composer.

As Fig. 1 shows, the approach needs several information to support its autonomy during execution. In particular, the Domain refers to a formalized *knowledge* related to the specific application domain where a service composition takes place. It is an ensemble of (1) concepts and relations (ontology) that enrich service descriptions and (2) causal constraints (e.g. pre- and post- conditions) that cause services to be correctly ordered in a service composition. Domain rules, instead, represent higher-level knowledge that is useful to express some constraints that invalidate or validate the generated plans, so reducing the space of admissible solutions. Finally, context rules are used to observe the external world during execution in order to generate new knowledge that potentially can enrich domain ontology and rules.

This paper mainly focuses on automatic service composition in the context of autonomic workflow by presenting a tool able to generate concrete and runnable compositions starting from a repository of service descriptions, a domain ontology and constraints. In particular, the domain is expressed as WDSL service descriptions annotated with OWL-S ones, whereas the target compositions can be generated either in WS-BPEL or in XPDL.

The rest of the paper is organized as follows. Section II discusses the related work on tools for automatic service composition. Section III presents the process for automatic generation of service compositions, the proposed tool and its architecture. Section IV analyzes some examples of automatic service composition in the context of emergency handling. Finally, Section V concludes the paper.

II. Related Work

To generate an executable business process description starting from a general, and possibly formal, description of user business requirements and service domain is a complex problem, whose solution needs: (1) a support for formally describing service domain and business problems; (2) an efficient technique for finding a service or combinations of multiple services from the domain, satisfying the specified problem; (3) the automatic generation of a formal business process description, possibly in a standard language (e.g. BPEL, XPDL, etc.), from the abstract plan. In spite of the plethora of efforts devoted to theoretic aspects, up to now only a few proposals have addressed the problem in a comprehensive way and very few tools to generate an executable business process description exist.

In relation to the first problem, the OWL Web Ontology Language [2] allows for representing domain knowledge through a formal and shared XML-based specification of concepts and relations among them. OWL-S [3] supplies Web service providers with a core set of markup language constructs for describing properties and capabilities of their Web services in unambiguous, computer-interpretable form, by referencing concepts and properties from OWL ontologies.

SAWSDL [4] is another W3C recommendation for semantically describing services. It introduces a set of extension attributes to be directly used in WSDL service descriptions to semantically annotate WSDL elements. WSMO-lite [5] is a lightweight set of semantic service descriptions in RDFS for annotating various WSDL elements, using the SAWSDL annotation mechanism.

Regarding the second problem, several approaches, techniques and tools [6-8] have been proposed in literature to efficiently tackle the automation of the composition process. Many of the proposed approaches are based on the use of AI planning techniques, handling the Web service composition problem as a state-space, constraints satisfaction, situation-calculus or other kind of planning problems. Semantics is considered an important support for the automation of the composition process [9].

The Planning Domain Definition Language (PDDL) [10] is considered the *de-facto* standard for classical planning problems input languages. A PDDL planning problem is described in two sections: domain definition and problem specification. The domain describes all the elements which characterize the domain for planning, i.e.: object types, predicates, actions (by specifying their inputs, outputs, preconditions and effects), etc. The problem essentially describes initial and goal states, by specifying the set of predicates assumed to be true in the initial state and the set of predicates to be satisfied in the goal state. Several planners have been developed which use PDDL as input language.

SHOP2 [11], is an HTN (Hierarchical Task Networks) planner, which exploits hierarchical relations among tasks for composing Web services. These relations have to be provided in advance to the planner by designers when describing the planning domain. The planning problem is solved by translating its OWL-S description into a SHOP2 description and by converting the SHOP2 generated plan to an OWL-S runnable process. As pointed in [12], SHOP2 performs well where complete and detailed knowledge on at least partially hierarchically structured action execution patterns is available, but, when no concrete set of methods and decomposition rules are available, an HTN planner is not able to find the solution. This problem inherently limits the planning ability of an HTN planner to the availability of decomposition methods designed by human experts.

In [12,13], Klush et al. proposes the OWLS-Xplan planner, which combines graph-based (by using GraphPlan [14]) and HTN planning, using OWL-S descriptions (of both domain and problem) as input, translating them into an XML version of the PDDL language, called PDDXML. The output is a sequence of activity described in PDDXML. The approach combines both the advantages of task decomposition available with HTN planning and the GraphPlan capability of always finding a solution, when present. The authors also propose a replanning component, able to update plans during execution.

Another recent composition framework is PORSCE II [15]. Like OWLS-Xplan, the framework input consists of OWL-S service descriptions, which are translated into PDDL. The framework combines a domain-independent planning component (e.g. JPlan, LPG-td) and an ontology concept relevance module for semantic awareness and relaxation during planning. Several plans, with different semantic accuracy levels, can be generated and presented to the user through a graphical component. Moreover, the graphical component can be used to request replanning by selecting a task and asking the system to find an alternative equal or semantically similar service or composition. Subsume relationships among service pre- and post-conditions are considered to find such alternatives.

Other notable planning solutions for service composition are based on the Golog language. Golog is a logical programming language and has been extended in [16] to support customized constraints and non-determinism in sequential executions and have been used in order to support service composition, by means of a translation into PDDL. In [17], a process for translating OWL-S descriptions into situation calculus has been proposed, while, in [18], DL reasoning techniques are used together with extended Golog to calculate conditional Web service compositions.

The Haley framework [19, 20] includes a Golog-based planning system for Web service composition. The system uses SAWSDL semantically described services as input, contains a planning Golog-domain generator and the eDT-Golog planner. Differently from the previous described framework, Haley is able to generate a WS-BPEL description of the plan and execute it on a WS-BPEL engine. However, Haley tackles the service composition problem from the perspective of generating complete business processes from user business requirements, by assuming the presence of concrete services with specified QoS parameters. In this sense, scalability is a very important problem and the hierarchical approach, as in SHOP2, is a way to reduce the planning effort, but requires a designer to know how to decompose tasks in subtasks. From our perspective, planning has to be a support especially to the generation of small business sub-processes, which concretize tasks from an already defined main workflow and is guided by the Binder component of the Configurator (Fig. 1). Consequently, the scalability problem is reduced in our perspective. Moreover, our proposed composition tool is also able to work with already composite service. Generally speaking, Haley's authors believe classical planning techniques are not well suited to the Web service domain, because of its inner non-determinism. As presented in [1], we argue that non-determinism can be handled through events observation and proper reaction: this way, the autonomic approach can be exploited to fill the gap with the classical planning techniques in Web service composition. Another important difference between Haley and our composition tool is that Haley is not able to generate concurrent sub-processes.

Finally, in relation to the third problem, the generation of formal and standard business process representation of the service compositions, several languages have been proposed. Among these, Business Process Modeling Notation (BPMN) [21], Xml Process Definition Language (XPDL) [22] and Web Service Business Process Execution Language (WS-BPEL) [23] are the most important and widespread ones. In the following, we focus our attention on WS-BPEL (v. 2.0) which can be considered the de-facto standard for business process description languages in the Web Service domain.

III. Tool Description

The proposed tool is intended to support the initial definition (plan) of a service composition or to re-plan an already defined composition.

A. *Composition Process*

Fig. 2 shows a graphical representation of the notion of Web Service composition.

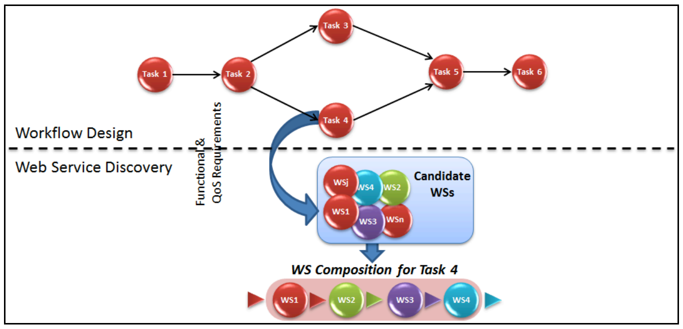

Figure 2. Web Service Composition Process

Starting from a task description (the problem, e.g. task 4 in the picture), a set of candidate Web Services (the service domain) is inspected in order to find a chain of services (a plan), which is consistent with the task description. Consistency means that, starting from a provided description of the initial state (i.e. the set of predicates which are true before the task beginning), the chain is able to reach the goal state (i.e. the predicates in the goal state have to be true at the end of the service chain).

The initial state includes a description of all the input data available at the beginning of the execution, while the goal state specifies the desired outputs to be obtained by the task execution. The problem task may correspond to a single task in an already defined business process or to the whole business process. The first case can be related to a re-planning process, where an activity of the process is replaced by a service composition, while the second case relates to the planning process, that is the definition of a new complete business process, satisfying some user's specific business needs.

We assume both the service domain and the problem to solve are described by using the OWL-S ontology, according to the IOPE semantics.

Any service operation is associated to an OWL-S file, which includes the definition of an OWL-S atomic process (*<process:AtomicProcess>*), specifying the inputs, the outputs, the preconditions and the effects (IOPE) of the specific operation performed by a service. Inputs and outputs may refer to concepts imported from OWL ontologies or XML-Schema data types. Preconditions and effects are specified within the OWL-S process description in the *<process:hasPrecondition>* and the *<process:hasResult>* sections respectively. Semantic Web Rule Language (SWRL) [24] expressions are used to define these conditions.

The problem is seen as a desired operation, specified in an OWL-S process with inputs, outputs, preconditions and effects, like a service operation. Inputs and preconditions make up the initial state, which is data known to be available and predicates known to be true when the related task starts, while outputs and effects make up the goal state, that is desired data outcomes and true predicates at the end of task execution.

Fig. 3 describes the main logic flow associated to our composition tool. The OWL-S service domain and problem descriptions represent the main inputs for the composer. Also, it is possible to specify simple business rules that have to be validated on the generated plans. Provided inputs are then processed and transformed into proper internal data structures.

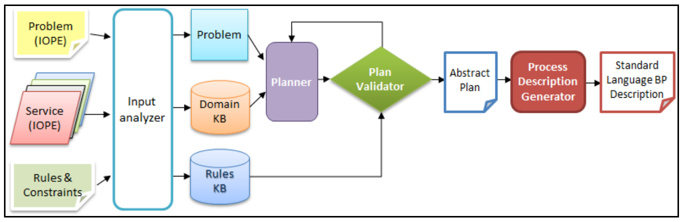

Figure 3. Planner-based composition process

The processed information (service domain and rules) is captured into internal incremental knowledge bases, used to quickly solve future requests related to the same domain or business rules. The internal input data are then supplied to the planner component to find some solution plan (set of activities) from the domain, satisfying the specified problem and business rules (through the plan validator). If the validation fails, a new plan may be found. The generated abstract plan is finally bound to concrete Web services and transformed into a standard business process representation (e.g. BPEL, XPDL). The produced description can be executed on execution engines, like RiftSaw [25], Apache ODE [26], SAWE [27].

B. Architecture and Implementation

The main composition process, described in the previous paragraph, has led to the tool architecture detailed in Fig. 6.

The most relevant components in the architecture are:

- The **OWL-S Analyzer**
- The **Planner**
- The **Plan Validator**
- The **Plan Converters** (or serializers)

In order to introduce flexibility in the proposed planning-based approach to composition, we have defined a **meta-model** for describing all concepts, and their relationships, which define our notion of autonomic workflow.

Fig.4 represents a Domain and a related Problem, while Fig. 5 gives a detailed description of the concept of Plan, composed of a Workflow, which includes simple or complex (composite) Activities and several control flow structures as Parallel and Sequence.

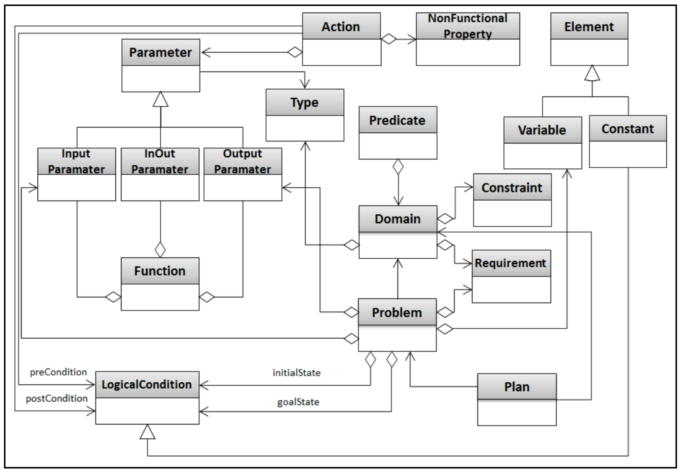

Figure 4. Problem and Domain Model

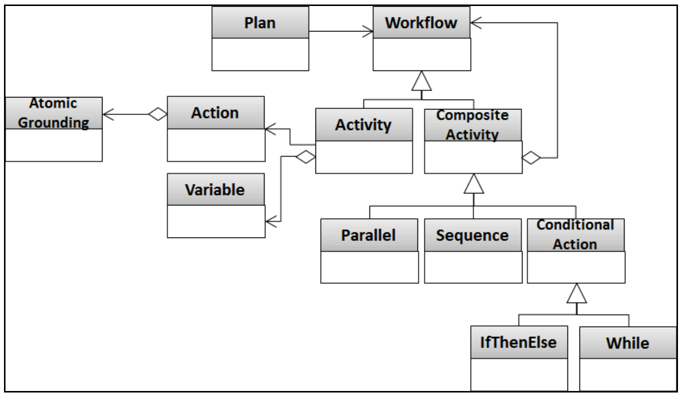

Figure 5. Workflow Model

The Plan element joins the two models.

The **meta-model** is used to define abstract representations of the service domain, the problem under analysis and workflow plans for solving it in the domain. By defining converters (problem and domain serializers) from the meta-model to planning specific representation languages, it is easy to support several planners.

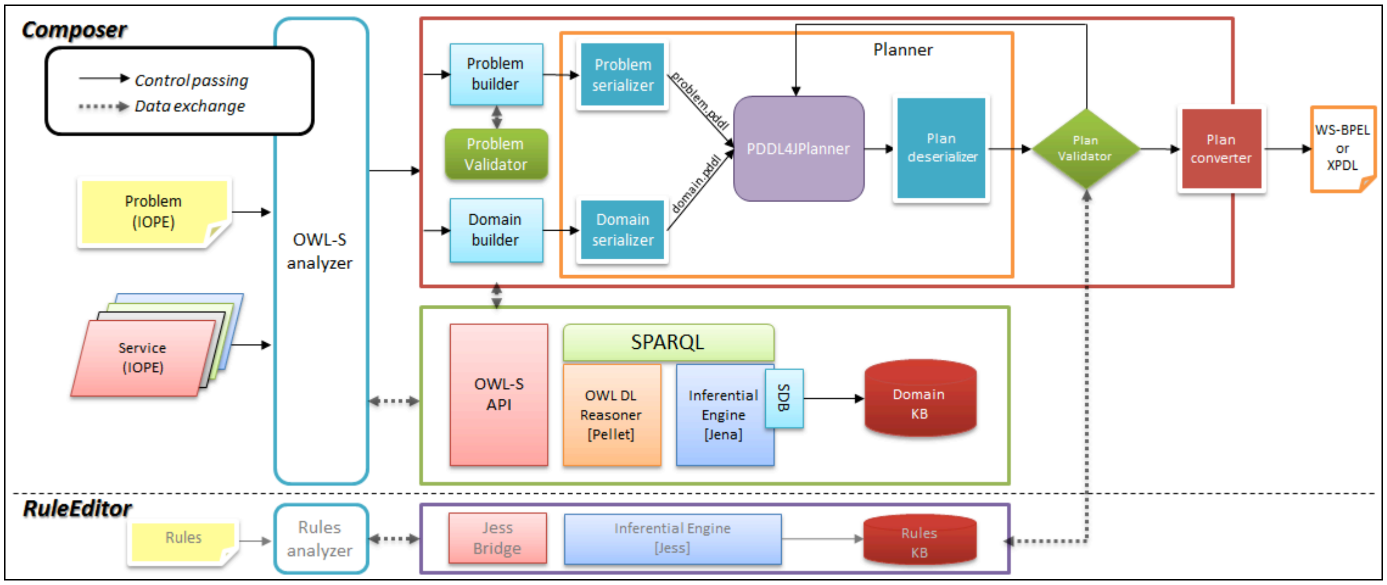

Figure 6. Detailed Composer Architecture

This way, the composer is independent from specific planning tools and languages. Since PDDL represents the de-facto standard for classical planning problems and there are several planning systems supporting this language, in the current implementation of our tool, we focused our attention on this language and implemented specific converters from the meta-model to the PDDL 3.0 specification.

The **OWL-S Analyzer** is the component responsible of analyzing both the OWL-S files, which describe the available services in the planning domain, and the OWL-S of the problem to solve. This component parses the provided inputs and converts them to an instance of the meta-model. More specifically, the main rules used by the analyzer to generate a meta-model instance from OWL-S files are the following:

- The service operation name (<*service:Service*>) defines the name of a new *Action*;
- The name of input and output parameters (<*process:Input*> and <*process:Output*>) of the atomic process describing the service operation defines the name of the action *Parameters*;
- The types of input and output parameters (<*process:parameterType*>), possibly referring ontology concepts, define new *Types* of the domain object and are associated to the corresponding Parameter objects.
- SWRL conditions, defining preconditions or effects of a service operation (<*process:hasPrecondition*> and <*process:hasEffect*>), are used to define domain *Predicates* and are associated to action objects as preconditions or effects respectively.
- Any action input parameter, retrieved from an OWL-S process, generates an *hasKnowledge* predicate, added as precondition for the relative action object and having the input parameter associated as a predicate variable.
- Any action output parameter, retrieved from an OWL-S process, generates an *hasKnowledge* predicate, added as effect for the relative action object and having the input parameter associated as a predicate variable.
- Any element in the OWL-S description, different from input or output parameter, or type names, defines a new domain *Constant*.
- References to WSDL information included into the OWL-S descriptions (<*grounding: WsdlAtomicProcessGrounding*>) have to be stored in an *AtomicGrounding* associated to the action object.

In a very similar manner, it is possible to build the meta-model *Problem* object, by applying the previous specified parsing rules to the problem OWL-S description.

The **Planner** is the component deputed to the processing of domain and problem inputs in order to produce a plan of domain actions satisfying the problem. To this end, several solutions are available. We implemented a PDDL problem serializer, which take as input the meta-model representation of the problem and produce a PDDL 3.0 compliant serialization of it. Similarly, we implemented a PDDL domain serializer, which does the same on the domain meta-model object built by the OWL-S Analyzer. By having the PDDL representation of both the domain and the problem, a PDDL planner can be used for generating plans.

The current implementation of the tool uses PDDL4J [28] for planning, a product released under the GNU General Public License (GPL), which is based on a Java implementation of the GraphPlan algorithm. This algorithm evaluates IOPE dependencies among actions and generates, in a finite number of steps, a *partially ordered plan* satisfying the goal from the specified initial state, if a solution is present in the domain. In the worst case, the GraphPlan state-space search procedure has an exponential complexity.

The PDDL4J output plan is represented in a PDDL-like representation, which is converted back to its meta-model representation (the Plan object in Fig. 5), through the *PlanDeserializer* component.

The grammar described in Table 1 defines the language supported by the Composer to express business rules that has to be satisfied by the generated plans. In the current implementation, the rule language only supports the definition of dependency (`<->` in the table) and mutual exclusion constraints (`->!`) between pairs of activity.

Table 1. An excerpt of the rule language grammar

```
EXP ::= EXP_TYPE ; | ;
EXP_TYPE ::= RULE | CONSTRAINT
CONSTRAINT ::= ACTIVITY <-> ACTIVITY |
               ACTIVITY ->! ACTIVITY
ACTIVITY ::= IDENTIFIER
RULE ::= …
```

As an example of these constraints, rule `A <-> B` means that if the plan contains the `A` activity, `B` has to be present too and vice-versa (whereas the order is inferred by IOPE descriptions). Rule `A ->! B` means that `A` and `B` activities have never to be both present in the same plan.

The **PlanValidator** component has the role to check if the business rules, specified by the user as input to the composer, are satisfied by a plan produced by the Planner. If these rules are present and the plan does not satisfy them, a new plan has to be searched by the Planner component until rules are satisfied or there are no other feasible plans.

The behavior of the composer, in relation to the described components, is shown in the UML sequence diagram of Fig. 7. In order to execute a plan generated by the planner engine on common business process execution engines, it is necessary to convert its meta-model representation (the *Plan* object) to a standard business process representation language (e.g. WS-BPEL, XPDL, etc.).

To this purpose, according to the architecture depicted in Fig. 6, several **Plan Converters** can be defined, which serialize the meta-model *Plan* object to a specific language representation. In particular, the current implementation of the tool has been equipped with two serializers: a *WS-BPEL Serializer* and an *XPDL Serializer* (Fig. 8).

A serializer for a business process representation language is responsible to retrieve all the information required by the corresponding language specification, and possibly demanded by the business process execution engine, to generate a compliant representation of the plan, executable on that engine. For example, the BPEL representation of a process to be executed on the RiftSaw business process engine is composed of a *.bpel* file (*BPELProcessFile* in Fig. 8), a *.wsdl* artifacts file (*BPELArtifactsFile*), a deploy *.xml* file (*BPELDeployFile*) and the set of imported *.wsdl* file.

The classes in Fig. 8 have been designed to maintain all the information required to correctly build the files. They provide a *serialize* method that can be invoked to generate the corresponding files, producing the serialization of the BPEL process.

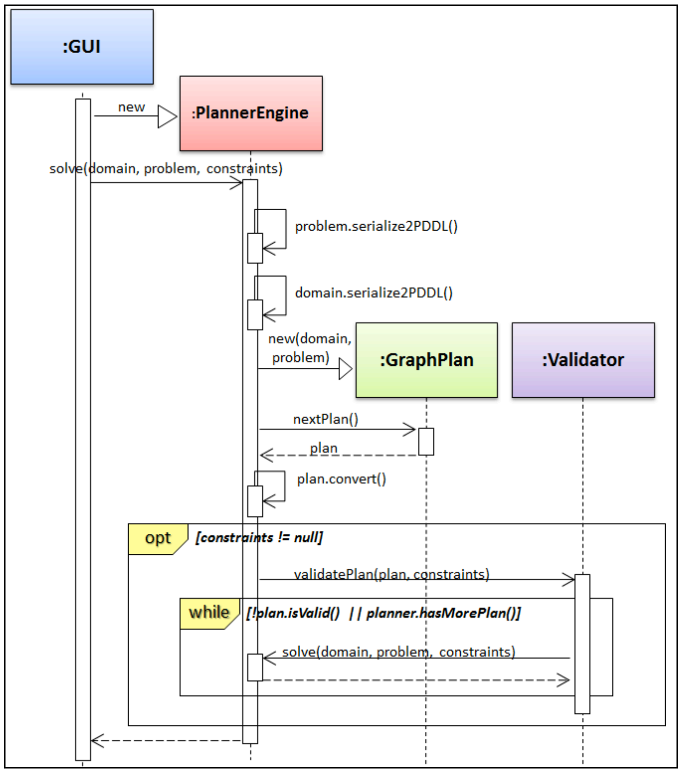

Figure 7. Plan generation and validation

The *BPELCreatorImpl* class is responsible of creating the *BPELProcessDefinition* object (and the component objects), retrieving all the necessary details from: 1) the *Plan* metamodel object; 2) the domain/problem OWL-S files; 3) the WSDL service files, which contains the endpoints required to make executable the final business process representation. The association between the actions of the plan, the OWL-S and WSDL files is retrieved from the OWL-S grounding during the OWL-S parsing and is stored in the Grounding information, provided for the Action class in the meta-model (Fig. 6). The *BPELCreatorImpl* is also responsible of generating the BPEL control flow associated to the plan (a partially ordered plan, as in the case of the GraphPlan engine used in the current implementation of the tool, can only generates sequences and parallel flows) and building a proper activity data flow, when data are exchanged among the actions of the plan.

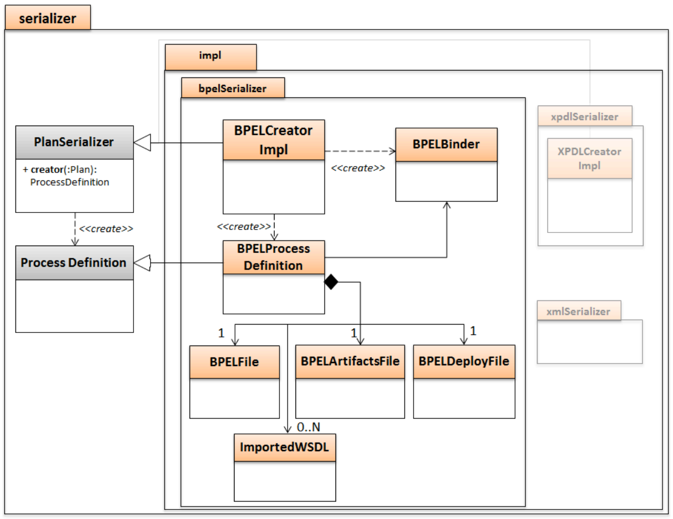

Figure 8. Plan serialization to a business process representation language

IV. An Application Scenario

To evaluate the potential of the tool, we tested automatic service composition on a specific application scenario implemented in SIEGE, a research project co-funded by Italian Ministry of Research [29], which adopted the tool to build specific workflows of services in the e-Government context.

The scenario is about the handling of a hydrogeological disaster (e.g. extreme rain, flooding, inundations, etc.) that strikes a human community (cities, towns, rural villages, etc…). In such a situation, the tool can be used to plan emergency management flows of actions to be executed by a workflow engine (alerting the population, scheduling support for evacuation of elderly people, coordinate resources to be used in rescue operations may be relevant examples)

Disaster response management is a particularly meaningful test bed for our tool since action flows must be timely planned and executed. Such actions may be concerned with the use of specific resources (such as telecommunication facilities) and/or the coordination of static and mobile resources (volunteers, policemen, ambulances).

Planning has to take in account resources capabilities, availability and readiness. To this end, the proposed tool is able to generate, automatically, quickly and almost effortlessly, all the planning processes needed. E.g. variations of the services available in the domain, changes in the state of resources, changes in the overall goal, can be immediately considered to get updated plans.

The specific example is about population alert by using multiple communication channels (mobile networks, SMS, MMS, automatic calls, TV, etc.). Alerting citizens is generally extremely important to reduce damages and harm to the population. By using targeted and informative alert messages, people can put in place their own precautions and better efforts to protect themselves and their families.

To design a realistic scenario, we have derived it from the analysis of real disaster management plans defined by the organization and emergency procedures of the Italian *Protezione Civile*, the national body in charge of prevention and management of disaster events. The *Protezione Civile* adopts a specific model ("metodo Augustus"), which emphasizes operational flexibility by using to the largest possible extent resources located close to the emergency, possibly belonging to different cooperating organizations.

The assumption is that such cooperating organizations would have made some or all of their disaster management resources or capabilities available as web services. Such web services could be used to access resources, to get info, to alert volunteers. Each service would have a WSDL and an OWL-S semantic description of its behavior. OWL-S description refers to a general domain ontology that describes the emergency context.

We considered organizations like local police, fire brigade, telecom companies, white pages, register offices, etc. For each one of these organizations, we defined a specific OWL domain ontology to classify the concepts involved and their relationships. The whole set of domain services refers to several dozens of entities. Some concepts included in the ontology are: *citizen*, *address*, *personal data*, *message*, *MSISDN* (mobile number), *deliver status of a message*, etc. The semantic OWL-S descriptions of each Web service refers to concepts described in the defined domain ontology. Some example of the domain services are:

- *register office service*, to get personal data of the people that must be warned;
- *white pages service*, to get personal data of people that could be involved in the alert process;
- *SMS/MMS broadcast* (telecom companies) to alert people in a specific neighborhood;
- *SMS/MMS send* to specific people directory;
- *send phone calls*, using prerecorded messages or via human call center (telecom companies);
- *alert local police or fire brigade*,

In Fig. 9, we report a representation of OWL-S ontology for the "Send SMS" service, which is the archetype of a service made available by a mobile telecom company to send SMS to a list of users.

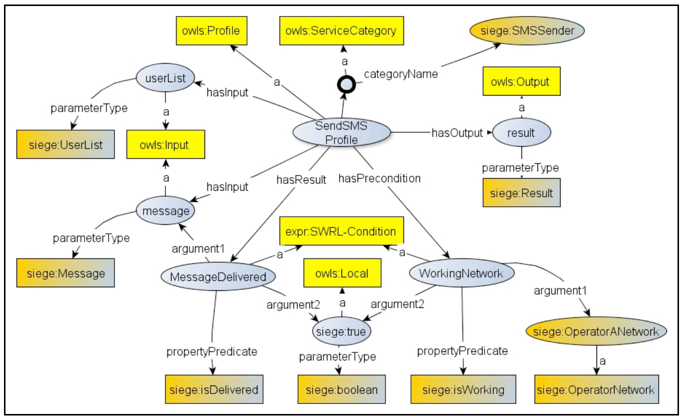

Figure 9. Ontological description of the service SendSMS

Such an OWL-S description is converted into PDDL, the internal language used by the PDDL4J planner.

In Table 2, we report a sample of PDDL translation:

Table 2. PDDL code generated by the PDDL Domain serializer

```
(:action SendSMS
    :parameters (?userList ?message ?result)
    :precondition (and (hasKnowledge ?userList)
                       (hasKnowledge ?message)
                       (isWorking OperatorANetwork))
    :effect       (and (hasKnowledge ?result)
                  (isDelivered ?message))
)
```

By using the backward strategy of the GraphPlan algorithm, if the problem goal is equal to (or contains in conjunction with other predicates) the effects of the SendSMS service, the PDDL4J planner may include the above PDDL action in the plan and try to reach the specified precondition through a single service, or a service chain, beginning with the specified initial state.

As the final goal of the whole scenario is about alerting the population living in a specific area, the goal includes the following post-conditions: to get home phone numbers of all people living in the area and to alert them by phone calls, by SMS, using TV channel, etc.

The resulting plan process includes several services and three different parallel branches. The three branches contain the following actions:

1. Alert using land line phones:
   - Get the list of home telephones in the area using a White Pages service;
   - Get the phone numbers, give such phone numbers to a call center for automatic call procedure;
   - Get a list of citizens without home phone;
   - Send the list of citizens that were not warned (as they do not have home phone or because they did not answered the call) to local police.
2. Broadcast an alert message using TV specific service.
3. Alert using SMS channel:
   - Using a service by the birth register to get the names of all people living in the area;
   - Get mobile numbers;
   - Send SMS to everyone.

In Fig. 10, the schema of the solution plan is reported.

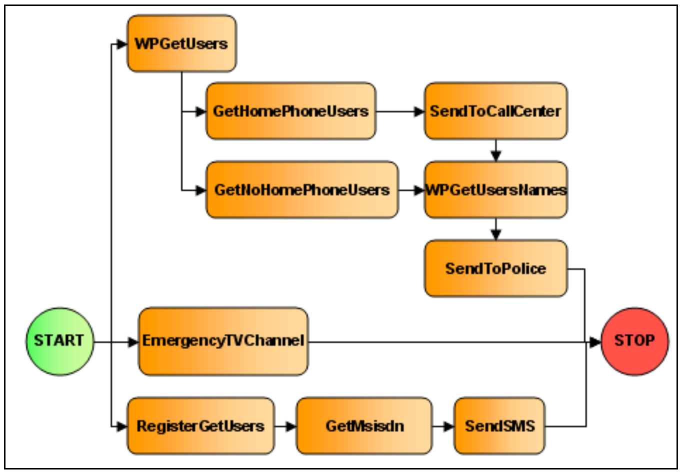

Figure 10. Schema of the plan for the alert workflow

V. Conclusion and Future Works

This paper proposes a tool for automatic composition of OWL-S semantically annotated Web services. Executable compositions are automatically generated by the tool in either WS-BPEL or XPDL languages, referencing concrete WSDL services retrieved from the OWL-S groundings.

OWL-S descriptions of both services and problem are analyzed, converted into PDDL and fed as input to the GraphPlan planner PDDL4J. Solution plans are translated into WS-BPEL or XPDL and can be executed by any common business process execution engine. An application scenario about the partial handling of hydrogeological disasters has been defined and used for testing the potential of the presented tool.

Instead of optimizing the tool for planning complete workflows, we have chosen to take advantage of it for re-planning (parts of) already defined (autonomic) workflows, when some activities of the original plan are not available and equivalent sub-processes are required to replace them.

Despite of its robustness and usefulness, the tool is still prototypal and further improvements are possible. Among these, current domain services are provided as input by means of a set of known OWL-S descriptions, instead we have planned to introduce more flexibility by integrating the tool with a registry able to automatically retrieve a set of candidate domain services matching the ontology concepts referred within the OWL-S problem specification. The registry can also be used for retrieving groundings of semantic services to concrete services to be referenced in XPDL or WS-BPEL descriptions.

Moreover, during XPDL or WS-BPEL generation, concrete services may ground same ontological concepts to different concrete data types, generating data mismatches when they are included within the same solution plan and present data dependencies over the differently grounded concepts. A possible solution consists of providing a set of Web service data adapters to the composer, to perform data conversions between different representations: they can be automatically selected and interposed between the mismatching services. Finally, when no semantically exact solution is available in the service domain, semantically relaxed plans for partial goal satisfaction could be proposed and ranked.

Acknowledgements

We would like to thank Roberto Pratola and Ivano De Furio for their contributions in implementing and validating the tool.

References

[1] G. Tretola, E. Zimeo, "Autonomic Internet-scale Workflows", in Proceedings of the 3rd International Workshop on Monitoring Adaptation and Beyond, ACM New York, USA, 2010,

[2] D. L. McGuinness, F. van Harmelen “OWL: Ontology Web Language,” W3C Recommendation, [Online] http://www.w3.org/TR/owl-features/, 2004.

[3] D. Martin, M. Burstein, and J. Hobbs et al, “OWL-S: Semantic Markup for Web Services,” W3C Member Submission, [Online] http://www.w3.org/Submission/OWL-S/, 2004.

[4] J. Farrell, H. Lausen, “Semantic Annotations for WSDL and XML Schema,” W3C Recommendation, [Online] http://www.w3.org/TR/sawsdl/, 2007.

[5] D. Fensel, F. Fischer, and J. Kopecký et al “WSMO-Lite: Lightweight Semantic Descriptions for Services on the Web,” W3C Member Submission, [Online] http://www.w3.org/Submission/2010/SUBM-WSMO-Lite-20100823/, 2010.

[6] J. Rao and X. Su, “A Survey of Automated Web Service Composition Methods,” in Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition, SWSWPC 2004, San Diego, California, USA, July 6th, 2004.

[7] S. Dutsdar and W. Schreiner, “A Survey on Web Service Composition,” in International Journal of Web and Grid Services, vol. 1, pp. 1-30, August 2005.

[8] Z. Li, L. O'Brien, J. Keung and X.i Xu, “Effort-Oriented Classification Matrix of Web Service Composition,” in Proceedings of the 2010 Fifth International Conference on Internet and Web Applications and Services (ICIW), pp. 357 - 362 , May 2010.

[9] B. Medjahed, A. Bouguettaya, A. K. Elmagarmid, “Composing Web services on the Semantic Web,” in the VLDB Journal vol. 12 no. 4, November 2003.

[10] M. Ghallab, A. Howe, C. Knoblock, and D. McDermott, et al, “PDDL: The Planning Domain Definition Language,” in AIPS98 planning committee (1998), Volume: 78, Issue: 4, Publisher: Citeseer, Pages: 1-27 DOI: 10.2307/3729517.

[11] D. S. Nau, T. C. Au, O. Ilghami, U. Kuter, J. W. Murdock, D. Wu and F. Yaman “SHOP2: An HTN Planning System,” in Journal of Artificial Intelligence Research, Vol. 20, pp.379-404, 2003.

[12] M. Klusch, A. Gerber “Semantic web service composition planning with OWLS-XPlan,” in Proceedings of the 1st Int. AAAI Fall Symposium on Agents and the Semantic Web, 2005.

[13] M. Klusch, K. U. Renner, “Fast Dynamic Re-Planning of Composite OWL-S Services,” in Proceedings of 2nd IEEE Intl Workshop on Service Composition (SerComp), IEEE CS Press, Hongkong, China, 2006.

[14] A. Blum, M. Furst, “Fast Planning Through Planning Graph Analysis,” Journal of Artificial Intelligence, vol. 90, pp. 281-300, 1997.

[15] O. Hatzi, D. Vrakas, N. Bassiliades, D. Anagnostopoulos, I. Vlahavas, “Semantic Awareness in Automated Web Service Composition through Planning,” in 6th Hellenic Conference on Artificial Intelligence (SETN 2010), Springer, LNCS Vol. 6040, Athens, Greece, May 2010.

[16] S. A. McIlraith, T. C. Son, “Adapting golog for composition of semantic web services,” D. Fensel, F. Giunchiglia, D. L. McGuinness, and M.-A. Williams, editors, KR, pp. 482-496, 2002.

[17] M. Phan, F. Hattori, “Automatic web service composition using ConGolog,” in ICDCS Workshops, p. 17, 2006.

[18] F. Lecue, A. Leger, A. Delteil, “DL Reasoning and AI Planning for Web Service Composition,” in Web Intelligence IEEE 2008, pp. 445-453, 2008.

[19] H. Zhao, P. Doshi, “Haley: An End-to-End, Scalable Web Service Composition Tool: A Hierarchical Framework for Logical Composition of Web Services,” in Proceedings of IEEE International Conference on Web Services, ICWS07, Salt Lake City, Utah, 2007.

[20] H. Zhao and P. Doshi, “Haley, A Hierarchical Framework for Logical Composition of Web Services,” in Service Oriented Computing and Applications, pp. 285-306, 2009.

[21] OMG, “Business Process Modeling Notation (BPMN) Specification v. 2.0,”, [online] http://www.omg.org/spec/BPMN], 2011.

[22] WfMC, “XML Process Definition Language (XPDL) v. 2.1”, [online] http://www.wfmc.org/xpdl.html, 2008.

[23] OASIS Standard, “Web Service Business Process Execution Language (WS-BPEL) Specification 2.0,” [online] http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html], 2007.

[24] I. Horrocks, P. F. Patel-Schneider, and H. Boley et al, “SWRL: A Semantic Web Rule Language Combining OWL and RuleML,” W3C Member Submission, [online] http://www.w3.org/Submission/SWRL/, 2004.

[25] Jboss Community, http://www.jboss.org/riftsaw, v. 2.3.0, 2011.

[26] The Apache Software Foundation, http://ode.apache.org/, v. 1.3.5, 2011.

[27] M. Polese, G. Tretola, E. Zimeo, “Self-Adaptive Management of Web Processes,” in Proceedings of the IEEE International Conference on Web Systems Evolution (WSE), 2010.

[28] D. Pellier, “PDDL4J”, [Online] http://sourceforge.net/projects/pdd4j/, 2011.

[29] S. Pierno, L. Romano, L. Capuano, M. Magaldi, L. Bevilacqua, “Software Innovation for E-Government Expansion,” in Move to Meaningful Internet Systems: OTM 2008, pp. 822-832, 2008.