Composing Semantic Web Services Online and an Evaluation Framework

George Markou
Dept. of Applied Informatics
University of Macedonia
Thessaloniki, Greece
e-mail: gmarkou@uom.gr

Ioannis Refanidis
Dept. of Applied Informatics
University of Macedonia
Thessaloniki, Greece
e-mail: yrefanid@uom.gr

Abstract — **This article presents an approach for semantic web service composition using Artificial Intelligence planning techniques. Our research prototype, MADSWAN, is able to support various stages of web service composition, both manually and automatically. It comprises a semantic web service registry, an editor of semantic description files, as well as manual and automatic web service composition modules. The system adheres to the reusable nature of web services, by utilizing existing open source projects as sub-elements. Furthermore, it tackles the problem's inherent non-determinism through the use of planning techniques, particularly contingency planning. Finally, we designed a set of evaluation benchmarks for web service composition systems, based on an existing collection of semantically annotated web services, and applied it in part to MADSWAN with very encouraging results.**

Keywords - Web service composition, non-determinism, OWL-S.

I. INTRODUCTION

In [1], we presented our ideas in regard to a system aiming to automate web service composition procedures through the combination of semantic web technologies and Artificial Intelligence (AI) planning techniques. Here, we extend this work, presenting our research prototype, **MADSWAN** (an anagram of "**M**anual **AND** **A**utomatic **S**emantic **W**SC"), a general evaluation benchmark set for web service composition systems, as well as some experimental results concerning the efficiency and effectiveness of **MADSWAN**.

The main goal of the semantic web [2] is to offer unambiguous and computer interpretable markup of the web's content, as well as of its properties and relations, using a language that has explicit, well-defined semantics [3]. This makes it possible to automate tasks that could previously only be performed by humans.

Web services, a major ingredient of the Semantic Web, aim to solve interoperability problems between heterogeneous systems, with transparency over the underlying technologies used to implement them and the platforms they are based on. This aim is facilitated by the semantic markup of web services in a language such as OWL-S [4]. OWL-S presents what a service does, how it is used and the effects it has, thus enabling the automation of tasks such as web service discovery, selection and composition.

Since an atomic web service does not often provide the desired functionalities on its own, it is necessary to perform the task of Web Service Composition (WSC) in order to achieve them. However, web services exist and operate in an ever-changing and expanding environment. For that reason, searching for the appropriate web services to achieve each goal is not an easy task; what makes WSC difficult and time-consuming is the additional burden of manually monitoring whether a web service taking part in an existing solution is still active and has the same usage and interface.

The problem of automatic WSC has been shown to be computationally hard, most recently in [5, 6]; in [5], it is stated that if the WSC problem is formulated as a composition of finite state machines, each one representing a web service, then in its simplest setting, when the composition is fully asynchronous, there is an *EXP-hard* lower bound on the complexity of the task. In [6], it is shown that solving the composition problem of non-deterministic web services with complete information is *EXP-hard*.

In a deterministic setting, we are forced to assume that we can predict the results of the executed actions precisely. However, such a setting is often restrictive and not realistic, and the adoption of a non-deterministic assumption, i.e., that the outcome of a web service is not known a priori, allows us to compute more flexible plans. In our work, we assume that non-determinism is inherent in the WSC domain and that it is always possible for a web service's execution to be unsuccessful or have undesired effects.

The automatic WSC process includes the following phases: presentation (or advertisement) of a single service in a registry; translation between external and internal service specification languages for the domain; generation of a composition process model; and, finally, evaluation and execution of the output composite service [7]. **MADSWAN** currently supports the first four phases of the WSC process.

MADSWAN is based on open source software components that utilize the current web service standards, with the main goal of creating a platform that allows its quantitative evaluation and comparison to other WSC systems. A typical user can advertise a new web service in the online registry, as well as retrieve and edit the web services stored in it through the system's online interface. Moreover, it is possible to semi-automatically create workflows based on OWL-S control constructs and bind them to web service descriptions and concepts that are present in the online registry.

Most importantly, however, users are currently able to generate composition process models automatically, based on deterministic AI planning [8, 9]. AI planning is the task of coming up with a set of actions that will achieve a goal; our future goal is to support contingency planning [10]. Contingency planning anticipates the different outcomes of nondeterministic actions, plans for a subset or all the possible contingencies that could arise, and allows for the construction of a conditional plan that can be executed correctly despite those contingencies.

Following a translation of the original semantic WSC domain, described in OWL-S, to an AI planning one, defined in PPDDL [11], the translated problem is solved by an AI planner. The solution is then converted back to an OWL-S process model. The whole process can be evaluated against pre-defined use case scenarios and simple quantitative criteria, such as the number of atomic or composite web services included in the output composite one, as well as the total planning time required to reach a solution.

This article extends our previous work [1, 12] by providing a rigorous analysis of **MadSwan**, presenting a working prototype, as well as focusing on its evaluation process. We provide details about the modules of the system, that is, the registry, the XML editor, and the manual and automatic WSC modules; we also provide benchmarks to evaluate **MadSwan**, giving details with regard to the specific ontological concepts used by them. Finally, we evaluate the composite web service descriptions that are generated by **MadSwan** for the deterministic setting against two different planners, using four problems from two different domains.

MadSwan is, to the best of our knowledge, the first system of its kind with a publicly available prototype able to support various stages of the WSC process. In combination with the presentation of an algorithm aimed to tackle the non-determinism in the WSC domain and the provision of quantitative evaluation benchmarks, these constitute a unique set of features for such a system.

The rest of the article is organized as follows: Section II reviews related work; Section III presents technical details concerning the implementation of **MadSwan**, the translation process between the WSC domain description language and the planning domain description language. Section IV focuses on evaluation by describing the benchmarks that are introduced to be used as test cases for WSC systems and presents an experimental evaluation of **MadSwan.** It also showcases the manual WSC module that is used for the generation of workflows for the predefined scenarios. Section V concludes the article and poses directions for future work.

II. Related Work

This section reviews the related work concerning, firstly the relevant WSC approaches, including those making use of AI planning and those that utilize different techniques and, secondly, the evaluation of such systems.

A. Web Service Composition

AI planning is the most widespread approach used to tackle the WSC problem. However, a significant number of approaches using different methodologies exist; although these approaches cannot be directly compared to the one presented in this article, we will briefly refer to them and then focus on the ones making use of AI planning.

An example of a non-AI planning approach is presented in [13]; its authors present a semi-automatic approach to WSC, with the output composite web services specified as process schemas, and the atomic web services that comprise the composite ones being selected at runtime, based on non-functional constraints specified by the users. The presented system, CCAP, is based on three core services: coordination, context and event services that schedule and implement user-configured adaptations of web services at runtime. The approach is considerably different than ours, since it is not fully automated and is only based on the syntactic content of web services. CCAP only makes use of technologies such as XML [14] and UDDI [15], without taking into account the semantic matching capabilities that can be achieved by using ontologies and semantic specifications, such as OWL-S.

The authors of [16] developed ITACA (Integrated Toolbox for the Automatic Composition and Adaptation of web services), a toolbox that supports the composition of BPEL [17] services in order to generate adaptation contract specifications. The process is based on the automatic extraction of behavioral models from interface descriptions that can be defined in WSDL [18], Abstract BPEL (ABPEL) or Windows Workflow Foundation (WF) [19]. However, as the authors note, although the adaptation process is automated, the final contract specifications may require human intervention to successfully complete the WSC process. Additionally, as in [13], the web services used do not carry any semantic content.

A framework for composing pre-existing services and components that is based on ITACA is presented in [20]. DAMASCo (Discovery, Adaptation and Monitoring of Context-Aware Services and Components) has been implemented as a set of tools that constitute a framework integrated in ITACA. The authors acknowledge the need for semantic representation instead of only a syntactic one and use model transformation, context-awareness, semantic matchmaking, dependency analysis and fault tolerance in order to achieve the goals of discovering, adapting and monitoring the composition of web services.

One of the first, and most well-known, approaches that convert the original WSC problem to a planning one is presented in [21]. The proposed system, SHOP2, converts the web services' OWL-S process models to a SHOP2 domain, and the WSC problem to a compatible Hierarchical Task Network (HTN). SHOP2 plans for tasks in the same order in which they will be executed, allowing it to be aware of the current state of the world at each step. Despite this advantage, the approach is planner dependent and does not deal with the domain's non-determinism Thus, it is limited in comparison to more general approaches that translate the WSC problem to one compliant with PDDL [22].

Another approach that utilizes AI planning to solve the WSC problem at hand is followed in [23], treating the application of a web service as a belief update operation. The authors identify two special cases of WSC that are tractable; these cases allow for a compilation into planning under uncertainty and the subsequent use of a conformant planner, that is, Conformant-FF [24]. This approach does not make use of a standardized web service description language, and the planner is only given as input a PDDL-like description of the domain, modified to describe uncertainty about the initial state.

PDDL and OWL-S are the de facto planning language and the most widely used semantic description language, respectively. For that reason, several attempts exist that tackle the WSC problem by utilizing the two languages in conjunction. OWLS-Xplan [25] is among the most well-known approaches utilizing a translation between PDDL and OWL-S. It incorporates a tool that translates OWL-S descriptions to corresponding PDDL-like ones, and then a hybrid planner is called to solve the generated planning problem, combining guided local search with graph planning and a simple form of HTN decomposition.

A similar approach is adopted in [26], in which standard PDDL files are produced during the translation phase, and consequently any PDDL-compliant planner can be employed to obtain a solution to the WSC problem. In practice, the authors incorporate two alternative planners, JPlan [27] and LPG-td [28]. A module that allows for the replacement of a single service if a user manually selects it for substitution is presented, with non-determinism in the WSC domain not being taken into account.

The authors of [29] also present a conversion schema from OWL-S to PDDL, based on [25, 30]. The presented methodology does not ignore the non-determinism in the domain, and makes use of a modification of an existing PDDL planner (Simplanner, [31]) to tackle it, through interleaving planning and execution. The proposed system does not support a full-featured registry that allows, e.g., the addition of new web services or the removal of existing ones by its users.

To the best of our knowledge, there are currently no web-based systems supporting multiples phases of the WSC process available. YaWSA [32] provides a web-based interface that supports a WSC process; however, it is no longer available for public use and, most importantly, it provided no other capabilities related to different phases of WSC, such as a registry.

The authors of [33] present a system supporting multiple phases of WSC, including web service browsing, the creation of composite services, service flow execution, and the generation of OWL-S descriptions used to describe their common process pattern instances. These instances are used to bridge the gap between the users' requirements and the technical service descriptions, as the authors view OWL-S as insufficient and not abstract enough to achieve such a result on its own.

The system that is most closely related to **MadSwan** in terms of functionalities is the one implemented in the SUPER (Semantics Utilized for Process management within and between EnteRprises) project [34]. The major objective of SUPER was to bridge the gap between the business needs expressed by business people and the actual Information Technology (IT) infrastructures intended to support them, while also supporting in a more efficient way the reuse and automation of business processes. For this reason, it implements a semantic-based and context-aware framework platform that supports the management of business processes in a scalable manner, through the use of semantic web services' technologies.

The final platform includes modules for the automated discovery, substitution, composition and execution of business process implementations. Furthermore, three use case scenarios were developed for the needs of the project, all based on the telecoms domain, covering the fields of fixed telephony, traffic routing and the management of mobile environments.

Despite the different objective of SUPER project in comparison to **MadSwan**, it shares a lot of similarities with it. The system's interface was alike the one in the manual composition module presented here, using the BPMN [35] standard as one of its basic elements. More importantly, the two systems share a similar architecture, e.g., the inclusion of modules for the discovery, translation and composition of semantic web services, even though the underlying standard for the description of semantic web services is WSMO [36] in SUPER and OWL-S in our case.

The evaluation process of SUPER was conducted solely based on interviews with a sample set of the system's users expressing their view on criteria such as the completeness and support, e.g., in terms of tools, of the system, or on its reuse of open source software and standards and its overall correctness. In our case, the evaluation process is based on quantitative criteria. The most important difference, however, lies in the WSC approach; WSC is used in SUPER to refine relevant parts of a business process model by searching for partial replacements in a process model. That is, the produced composite output is not presented as a new semantic web service, as in the approach presented here, nor does it take into account any non-determinism in the domain.

B. Evaluation of Web Service Composition Systems

It is noteworthy that the literature on WSC systems suggests a gap in their evaluation process; although recently there have been a few exceptions, a plethora of approaches simply rely on qualitative criteria and/or a single case study to evaluate their methodology. More importantly, the relevant literature does not suggest a standard test bed for WSC systems [23], or even a standard collection of web services to be used.

In [37], a comparison of planning techniques for WSC is conducted, with the evaluation criteria being based only on qualitative criteria. Some of the criteria that the authors take into consideration are: whether the technique is domain independent as well as whether it supports partial observability and non-determinism or not; the standards to which it can be applied to, i.e., its applicability, and its support of concurrency in the execution of web services. The scalability of the approach is also evaluated, but without any mention to quantitative results; the authors simply evaluate it based on a critique of the algorithm used.

In [38], current WSC approaches are criticized for making use of easy to measure criteria, without incorporating more important requirements in their evaluation. Several testing challenges in relation to the effective evaluation of service-centric systems are reported in [39]. Among these challenges are the dynamicity and adaptiveness that is inherent in the output workflows that contain abstract services, due to the fact that they can be automatically bound to various concrete services during the execution of the workflow instances. Other reported challenges are the lack of control that is attributed to a web service being modified during its lifecycle, and the cost of testing that is related to invoking the actual web services.

In [40], it is concluded that web service testing is more challenging than testing traditional systems; these findings are consistent with [38], that is, it is the authors' opinion that the difficulty in testing web services is partly due to the dynamic nature of web services and the limited control over them, as well as not having access to their source code.

Several approaches for WSC are evaluated in [41]. Among those not referenced previously, three out of eleven do not provide any evaluation at all. The rest, e.g., [42], present quantitative experimental results, such as the time needed to achieve a solution; however, none refers to actual scenarios with specific goals that exhibit the system's functionalities. For example, in [43], another one of the approaches evaluated in [41], randomly generated problems are used, with the authors only providing information regarding the number of services and ontology concepts present in them. The authors of [44] present quantitative experimental results, along with details in relation to the machine that was used to run the experiments and the number of web services taking part in the tests. A non-standardized language is used to implement the composition model, namely VCL (Vienna Composition Language), without any mention to the structure or the goals of the test problems, so as to allow their replication and comparison to other systems, or to testify that they are non-trivial.

Of the approaches evaluated in [41], only [45] denotes the test set that was used for evaluation purposes, the 2009 Web Service Challenge dataset [46]. The actual benchmark domains used in [47] are specified, consisting of two of the problem files used in [21].

Although a large part of the recent approaches related to planning, such as [29, 48], either evaluate their methodology on case studies without referring to quantitative criteria, or not at all [49], the same is partially true for non-AI planning approaches as well. This is the case for [16], where it is mentioned that the system has been validated and evaluated in synthetic problems and real-world examples, such as a travel agency, or library management systems, without, however, any results being reported.

Only recently a few AI planning approaches, such as [23, 26, 50], provided quantitative criteria for evaluation. The most extensive evaluation is presented in [23]; two artificial benchmarks are provided, each tested with different encoding methods and planners, and various elements of the planning process, such as the planner's total runtime or the number of search states and actions in the output plans, are measured.

In [24], a single case study is presented, with a different number of web services participating in the WSC experiments, and measuring the preprocessing, transformation (from OWL-S to PDDL), and planning time required. One of the two available planners in the system is used and evaluated, and the atomic web services that comprise the final composite one are (mostly) hand-tailored by the authors, although entire domains of the OWL-S Service Retrieval Test Collection (OWL-S TC) [51] are used for the composition in general.

Kona et al. [50] also incorporates rigorous evaluation experiments. Specifically, a single use case scenario is presented, with three variations depending on whether the produced workflows are sequential, non-sequential, or conditional non-sequential, along with the Inputs/ Outputs/Preconditions/Effects (IOPEs) of the services that take part in the WSC process. The criteria used in the evaluation of the system are quantitative, i.e., the number of web services participating in the problem, the number of I/O parameters each web service had, and the preprocessing and query execution time needed to obtain a solution. The web services that take part in the composition comprise a customized version of the 2006 Web Service Challenge [52] test collection.

An extensive evaluation is available in [20] based on two case studies; an online booking system and a road information one. Three variations for each are used, with increasing size and complexity with respect to the number of interface descriptions. The total numbers of states, as well as the required time and percent of CPU used are measured to evaluate the discovery and adaptation of both case studies.

Finally, the approach in [13] not only evaluates the system's performance based on scalability and adaptability criteria, but also provides a usability study based on 41 users of different educational backgrounds who were asked to use the system and report their experience by answering a questionnaire. However, the approaches in [13, 20] cannot be directly compared to **MADSWAN** mainly due to the systems' compatibility with dissimilar to ours underlying standards and technologies, such as the use of a UDDI registry in [13] with the services being described in WSDL, or the interface descriptions in [20] being defined either in WSDL, BPEL or WF. Secondly, due to their significantly different goals and motivation; for example, in [13] it is assumed that any non-determinism in the execution of web services has already been described in user configured exceptions.

III. THE MADSWAN SYSTEM

This section presents **MADSWAN**, particularly the modules that comprise it, their functionalities, as well as the steps of a typical use case, that is, creating a WSC problem, converting it to a planning one, solving it and, finally, translating the output plan to an OWL-S description file.

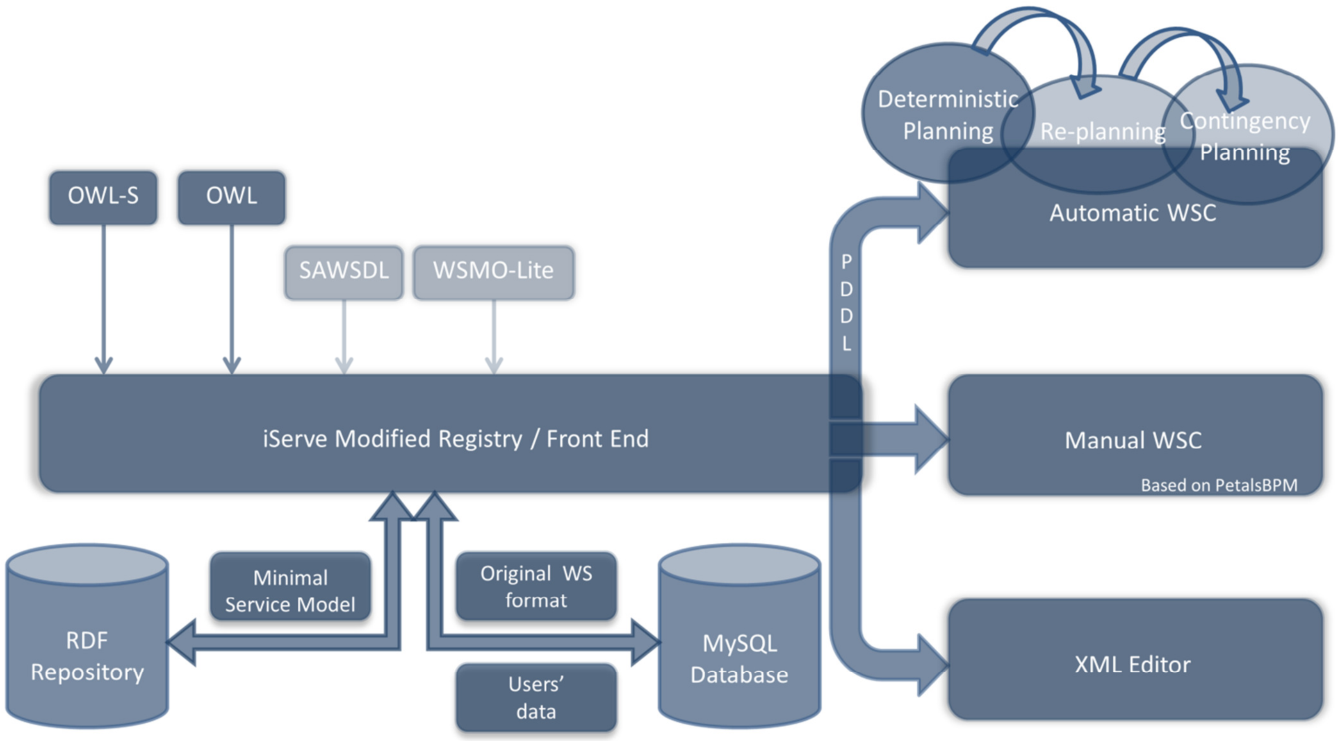


Figure 1. Architecture of **MADSWAN**.

A. *Available Functionalities*

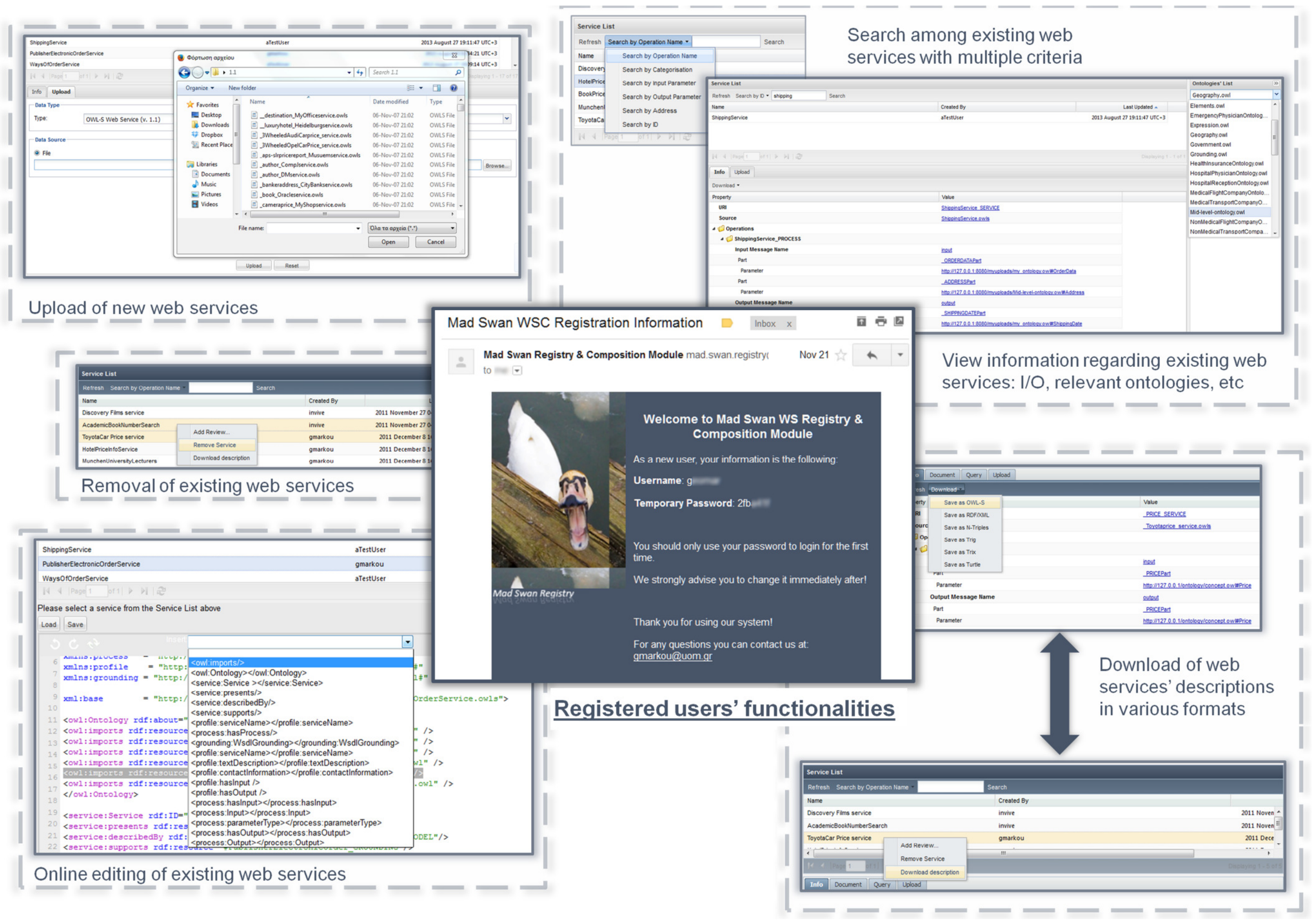
In our view, a WSC system should follow the same principles as web services themselves. Since web services rely on the idea of maximizing the reuse of loosely coupled components [53], our goal was to implement **MADSWAN** by making use of freely available components as much as possible. This led to a reduced required effort in comparison to creating entirely new components, and allowed us to use well established standards instead of proprietary ones. Moreover, such an approach facilitates the comparison of different WSC systems to each other.

MADSWAN supports various functionalities related to different stages of WSC, all of them being available through an online interface. An overview of the various components that comprise **MADSWAN** is shown in Fig. 1. The core of the system communicates with an RDF repository and an SQL database, in order to store the users' data and the inputted semantic web services and ontologies. These are currently in OWL-S / OWL format respectively, with the prospect to support other formats, such as SAWSDL, in the future. The three different modules that correspond to the manual WSC, the automatic WSC and the online web service editor are depicted on the right of Fig. 1, with the transparent components depicting functionalities that have not yet been integrated to the system.

The first functionality related to WSC is storing the service descriptions. In order to support semantic web service discovery in a more meaningful way, we decided against the use of UDDI, although it is one of the most well-known approaches for web service publication. UDDI's search mechanism is based on the description of the web services' capabilities using a classification schema that does not provide for a semantic description of their content. For this reason, instead of using UDDI, or approaches such as [54, 55] that bridge the gap between semantic web services and UDDI (in most cases between OWL-S and UDDI), we opted to use iServe [56] as the core of our application.

iServe is a service registry that supports importing service annotations in various formalisms, such as SAWSDL [57], WSMO-Lite [58], and OWL-S. This process is achieved through first transforming the original annotations to linked data, based on a common vocabulary for services, called "Minimal Service Model". Since iServe is open source, we created a modified version of its web-based application, making several improvements to the original registry's interface and functionality, and populating it with version 4.0 of the OWL-S TC. Fig. 2 illustrates the basic functionalities of the implemented registry through screenshots of the actual application. It shows the sub-components that are available for a registered user of the system, with unregistered ones being allowed to access only a subset of them, namely the ones that do not alter the registry's contents.

Fig. 2 illustrates that the registered users can upload new web service descriptions, which in our case are semantically expressed in OWL-S, and search among the existing web service descriptions based on criteria such as their operation name, input or outputs parameters, or IDs. Moreover, they can view information regarding them, among which are their


Upload of new web services
Search among existing web services with multiple criteria
View information regarding existing web services: I/O, relevant ontologies, etc
Removal of existing web services
Mad Swan WSC Registration Information
Registered users' functionalities
Download of web services' descriptions in various formats
Online editing of existing web services

Welcome to Mad Swan WS Registry & Composition Module

As a new user, your information is the following:

Username: [redacted]

Temporary Password: 2fb[redacted]

You should only use your password to login for the first time.

We strongly advise you to change it immediately after!

Thank you for using our system!

For any questions you can contact us at: gmarkou@uom.gr

Figure 2. Registered users' functionalities for the registry – System screenshots.

Uniform Resource Identifiers (URIs), or their inputs and outputs based on the ontologies already in the registry. They can also remove or update the descriptions of existing web services stored in the registry, or download their descriptions either in their original format, i.e., OWL-S, or in a variety of other formats, such as RDF/XML [59] or Turtle [60].

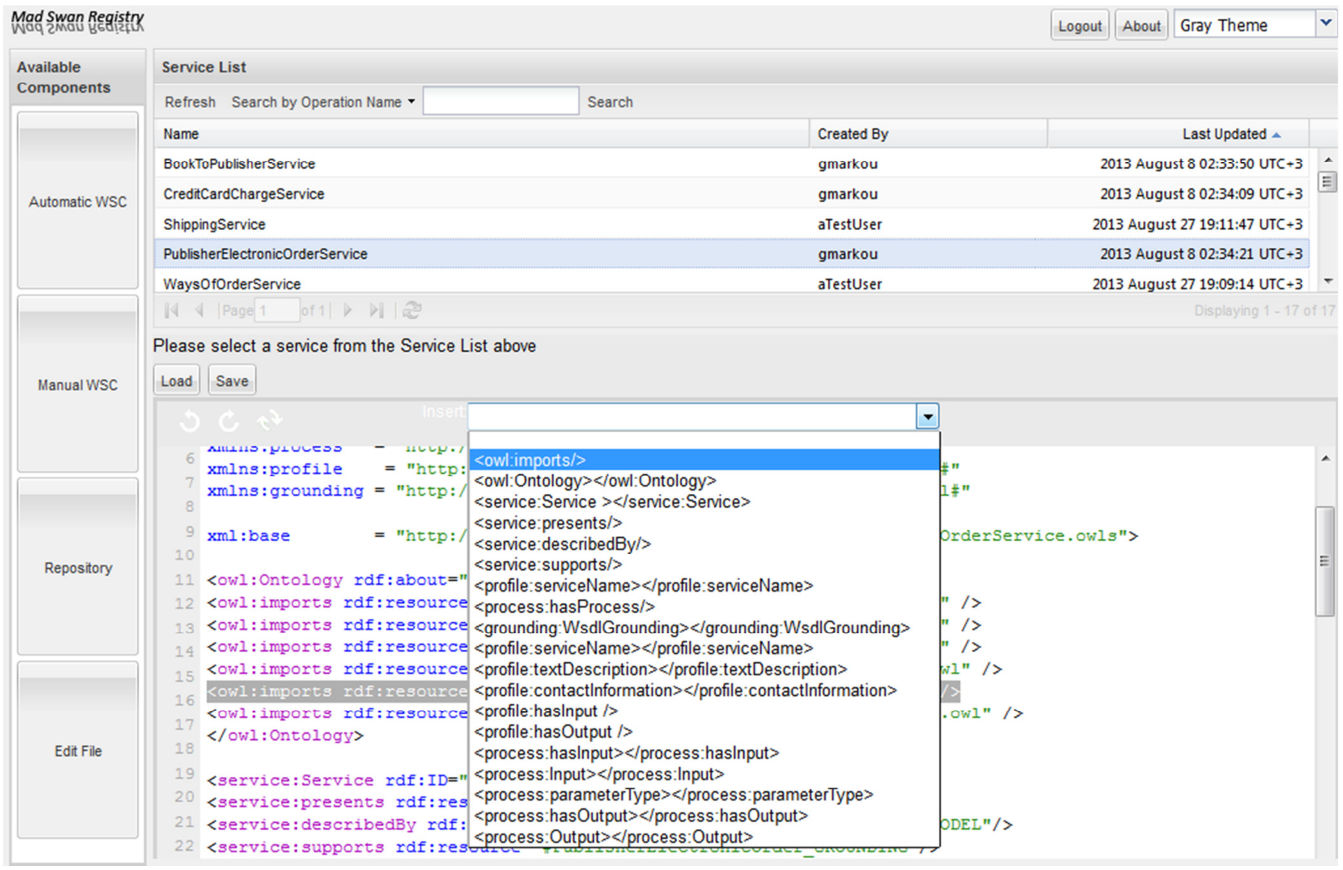
Finally, the users have access to an online XML editor, which uses syntax coloring to facilitate its use, as well as predefined templates with OWL-S syntax, so as to allow its use by non-expert users. Fig. 3 overviews the interface of the application along with the XML editor module. Specifically, the four basic sub-modules of the application appear on the left side of Fig 3. On the top, there is the service list containing the available web service descriptions of the registry, along with their uploader's username and the date when they were last updated. This part of the interface also contains the system's search functionality. Directly below is the online XML editor, where a web service description from the registry is loaded, and the user is ready to insert one of the available template OWL-S syntax expressions. More information and tutorials on the basic use of the system are available at [61].

For the purposes of the non-deterministic automatic WSC, we plan to use PPDDL, the planning language used in the non-deterministic tracks of the recent International Planning Competitions for the purposes of the non-deterministic automatic WSC. PPDDL is essentially a syntactic extension of PDDL 2.1, and supports modeling non-deterministic actions through probabilistic effects, which can be arbitrarily interleaved with conditional effects and universal quantification.

Since the web services in the registry are described semantically through OWL-S, a translation between the two languages must take place. There are various works that have proposed conversion schemas from OWL-S to PDDL that do not differ significantly from each other. As such, we also adopt an approach similar to [25, 26, 30, 62].

Our translation module is based on the source code of [26], with the necessary extensions to accommodate the creation of planning files that can handle non-deterministic actions, that is, generating PPDDL files instead of PDDL ones, in cases where the domain is non-deterministic.

Moreover, the planning problem file is created based on the users' choices; a user can choose between the ontology concepts that are present in the web services that take part in



Mad Swan Registry
Logout
About
Gray Theme
Available Components
Automatic WSC
Manual WSC
Repository
Edit File
Service List
Refresh
Search by Operation Name
Search
Page 1 of 1
Displaying 1 - 17 of 17
Please select a service from the Service List above
Load
Save
Insert
<owl:imports/>
<owl:Ontology></owl:Ontology>
<service:Service ></service:Service>
<service:presents/>
<service:describedBy/>
<service:supports/>
<profile:serviceName></profile:serviceName>
<process:hasProcess/>
<grounding:WsdlGrounding></grounding:WsdlGrounding>
<profile:serviceName></profile:serviceName>
<profile:textDescription></profile:textDescription>
<profile:contactInformation></profile:contactInformation>
<profile:hasInput />
<profile:hasOutput />
<process:hasInput></process:hasInput>
<process:Input></process:Input>
<process:parameterType></process:parameterType>
<process:hasOutput></process:hasOutput>
<process:Output></process:Output>

Name	Created By	Last Updated
BookToPublisherService	gmarkou	2013 August 8 02:33:50 UTC+3
CreditCardChargeService	gmarkou	2013 August 8 02:34:09 UTC+3
ShippingService	aTestUser	2013 August 27 19:11:47 UTC+3
PublisherElectronicOrderService	gmarkou	2013 August 8 02:34:21 UTC+3
WaysOfOrderService	aTestUser	2013 August 27 19:09:14 UTC+3

```
xmlns:process    = "http:/
xmlns:profile    = "http:                                    #"
xmlns:grounding  = "http:/                                   l#"

xml:base         = "http:/                                   OrderService.owls">

<owl:Ontology rdf:about="
<owl:imports rdf:resource                                    " />
<owl:imports rdf:resource                                    " />
<owl:imports rdf:resource                                    " />
<owl:imports rdf:resource                                    wl" />
<owl:imports rdf:resource                                    />
<owl:imports rdf:resource                                    .owl" />
</owl:Ontology>

<service:Service rdf:ID="
<service:presents rdf:res
<service:describedBy rdf:                                    ODEL"/>
<service:supports rdf:resource
```

Figure 3. Application's interface presenting its XML editor.

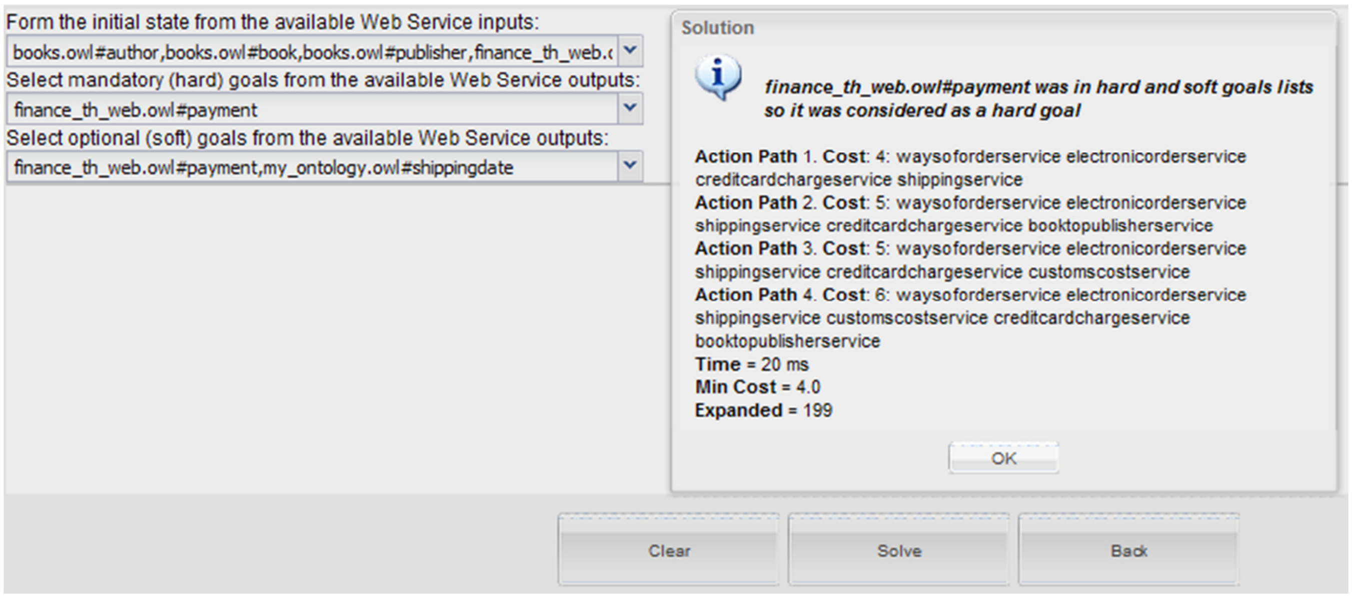
the composition (either a subset of the web services in the registry that he selected, or all of them). Then he should state which concepts formulate the initial problem state and the final composite web service's inputs, and which formulate the goal state and the final composite web service's outputs.

The goal state concepts are split in two lists, one containing hard and one soft goals, essentially "must-achieve" and "should-achieve" ontological concepts. In the case that a user has stated a concept to belong in both lists, then the system considers that concept to be a hard goal. Fig. 4 presents the system's interface for such a case; the four different plans, with increasing lengths, that were generated are depicted, along with the total time that was required to produce them, the length of the plan with the minimum cost, and the total number of states that were expanded during the plan search.

Examples of the currently generated planning files are available at [63]; these files consist of the planning domain and problem files for a random problem, the solution file that the planner outputs, and the OWL-S profile and process files that are the results of the translation process of the solution plan. A set of semantic web service descriptions that can be used to interact with the registry (taken from OWL-S TC and modified accordingly) are available at [64]. Although the development of **MadSwan** is still in progress, an alpha version of its online prototype is available at [65].

B. *Contingent Planning*

After the conversion of the OWL-S descriptions to planning domain and problem files, AI planning techniques can be used to generate the output plan/composite web service. In our view, the WSC problem is an inherently non-deterministic one. Indeed, it is always possible for a web service to be unavailable, or its execution to be unsuccessful or have undesired effects. For this reason, we adopt a non-deterministic formulation of the problem that allows us to compute more flexible plans. We opt for the incorporation of a contingent planner [66], in order to generate plans that can cope with the most influential and likely contingencies.
Our approach, which has not yet been integrated in the current version of the online prototype, is based on a complete search algorithm. This is used to generate all the possible plans for the most probable contingencies, starting from an optimal one, with an increasing cost, given a limited period of planning time. A - suboptimal - contingency plan can then be constructed by merging these plans. Merging is achieved through searching for natural join points, i.e., when search nodes share a predecessor through different sets of outcomes, and by removing any plans that contain redundant actions, that is, repetitive actions or ones that do not produce useful results.


Form the initial state from the available Web Service inputs:
books.owl#author,books.owl#book,books.owl#publisher,finance_th_web.o
Select mandatory (hard) goals from the available Web Service outputs:
finance_th_web.owl#payment
Select optional (soft) goals from the available Web Service outputs:
finance_th_web.owl#payment,my_ontology.owl#shippingdate

Solution

finance_th_web.owl#payment was in hard and soft goals lists so it was considered as a hard goal

Action Path 1. **Cost**: 4: waysoforderservice electronicorderservice creditcardchargeservice shippingservice
Action Path 2. **Cost**: 5: waysoforderservice electronicorderservice shippingservice creditcardchargeservice booktopublisherservice
Action Path 3. **Cost**: 5: waysoforderservice electronicorderservice shippingservice creditcardchargeservice customscostservice
Action Path 4. **Cost**: 6: waysoforderservice electronicorderservice shippingservice customscostservice creditcardchargeservice booktopublisherservice
Time = 20 ms
Min Cost = 4.0
Expanded = 199

Figure 4. Application's interface regarding the automatic WSC module and its solutions.

A somewhat similar approach using GraphPlan [67] is presented in [68]. Here, we opt for the use of A* [69], an optimal complete algorithm, with the use of the max heuristic (hmax, [70]) as an admissible heuristic function; however, this is not a restrictive choice. In practice, any complete algorithm can be used instead, in combination with a variety of admissible heuristic functions so as to produce optimal plans. The algorithm either outputs a contingent plan that covers all of the users' hard goals and any of the soft goals that have been set, or returns with a message that no plan was found in the allowed time period.

It is important to note that our approach does not try to develop a plan for every possible contingency, as the WSC domain may have too many sources of uncertainty for such a methodology to succeed. Since we cannot cope with every possible point of failure, a re-planning module will also be incorporated. As such, the approach is essentially offline, with a pre-computed contingent plan being used while the composite web service is executed. However, real time execution monitoring is essential, as the branches of the plan being used are determined by the actual outcomes of the atomic web services.

Re-planning occurs each time the contingent plan does not cover the current contingency, that is, an unexpected event occurs that is not already covered in the pre-computed plan. Such events may refer to a web service being unavailable at the time, or producing a result which is different than the one expected in the plan, e.g., a web service not being able to purchase a book due to it being out of stock.

Fig. 5 presents the aforementioned planning approach with an example. For reasons of brevity, we assume that the planning problem has three solutions. These solutions are of increasing length, the first being optimal with regard to the number of web services required to achieve the goal, requiring the execution of three actions (denoted by a rectangle), and the other two requiring an additional one.

The actions of the plans are color-coded, that is, the rectangles that share the same color are meant to depict the same action across different plans. We will refer to the actions by their respective color, e.g., the "green action". Each step in the creation of the plan is marked with its respective number on the right of the figure. Duplicate actions that are dropped from each plan and are not included in the final one are marked with a red "X".

In step 1, the contingent plan that we generate consists only of the base, optimal plan. In step 2, the second plan is added to the existing plan. Since the first two actions (white-purple ones) are common between the two plans, an alternative branch is created, consisting only of the two last actions (red-yellow). The new contingent plan is generated in step 3, and the next available deterministic plan is added to it in step 4. Since its first (white) and last (blue) actions are already present in the contingent plan, a new branch is added to it, consisting of the green and orange actions. The final contingent plan is shown in step 5, having three available paths to the goal, and for that reason, being able to withstand the occurrence of the same number of contingencies.

C. *From AI Plan to Composite Web Service*

Finally, we convert the plan back to an OWL-S (composite) web service, that is, we create an OWL-S profile and its process description, in a fashion similar to that described in [71]. In short, the resulting profile description file mainly refers to the new composite web service's IOPEs, as well as it defines some of its basic elements, such as its name or textual description. The output process model describes the plan that a client must execute in order to interact with the composite web service, and is based on OWL-S control constructs. The OWL-S API [72] that will be used to implement the conversion supports composite processes that use OWL-S control constructs, such as ⟨*Split+Join*⟩, and conditional constructs like ⟨*If-Then-Else*⟩, which is necessary to produce correct solutions to the use cases presented in Section IV.

Fig. 6 summarily illustrates our approach; it should be read as a timeline, starting from the left side of it. That is, OWL-S TC is used throughout all the stages of our application, whereas the OWL-S API is only used in the final stage, in the translation of the problem's solution to an OWL-S description file and the creation of the OWL-S profile and process files. Items in the same column imply that they are related to each other and occur in the same time frame.

D. *Manual Web Service Composition Module*

Since the relevant literature does not suggest a standard test bed for WSC systems, we decided to implement a manual WSC module so as to evaluate our automatic WSC approach against it. In order to create a manual OWL-S composer, we modified an existing open source application; Petals BPM [73] is a BPMN 2.0 modeler, which we adapted to accept the OWL-S constructs that are necessary for WSC. Moreover, we added a few "helper" constructs to provide a more intuitive interface.

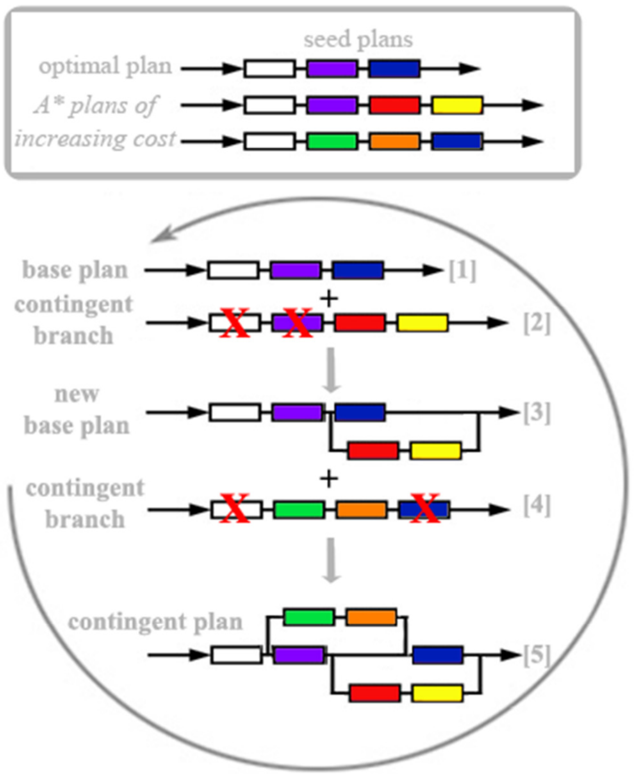


Figure 5. A graphic example of the planning algorithm.

The OWL-S constructs currently supported by the module are the ⟨*Sequence*⟩ (implicitly), ⟨*If-Then-Else*⟩, ⟨*Split+Join*⟩, and ⟨*Repeat-While*⟩ control constructs, along with the necessary inputs, outputs and web services' elements. The "helper" constructs comprise of an ⟨*End Split+Join*⟩ and an ⟨*End Repeat-While*⟩ construct, used

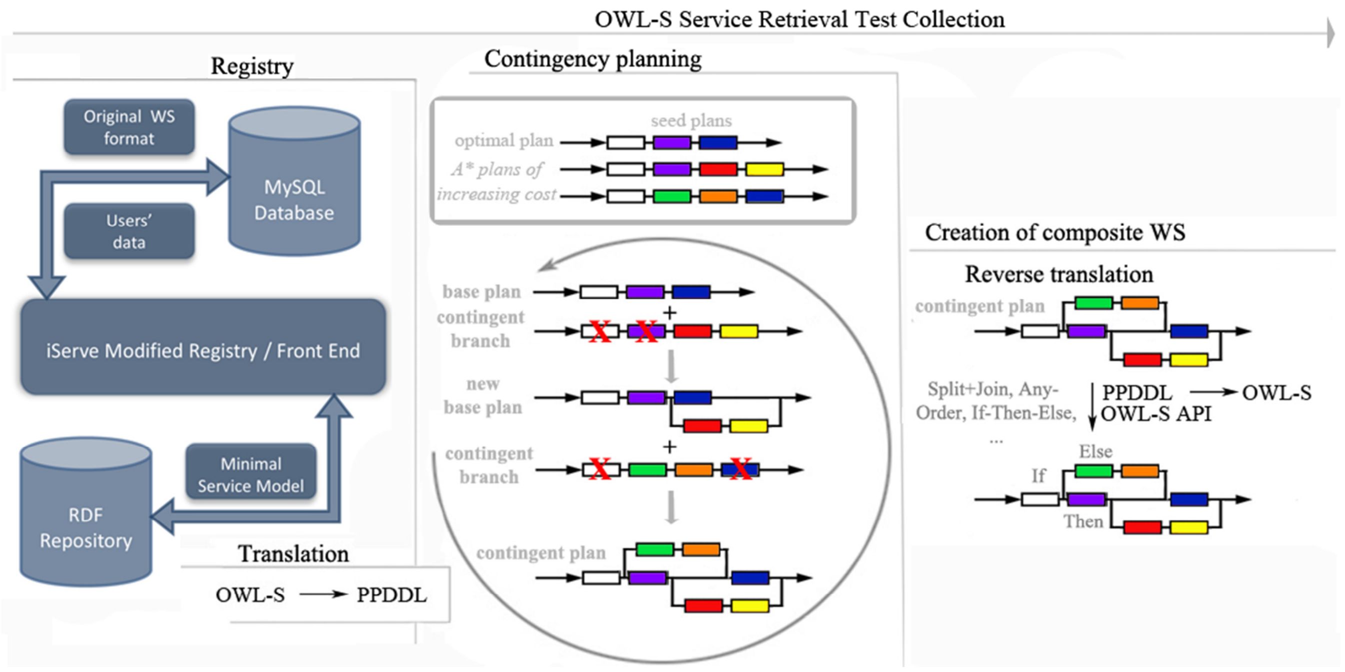


Figure 6. System overview timeline.

in conjunction with the regular ⟨*Split+Join*⟩ and ⟨*Repeat-While*⟩ constructs to enclose other elements in them, and dedicated ⟨*If*⟩ and ⟨*Else*⟩ sequence flows that are only used along with an ⟨*If-Then-Else*⟩ gateway. Moreover, there are ⟨*Start*⟩ and ⟨*End*⟩ constructs to signify the beginning and end of a workflow. Finally, users can bind the web service and data input/output constructs that they added to the workflows to specific web services and relevant ontologies' concepts already present in the registry, respectively.

It should be noted, however, that having bound such specific concepts from the registry to the available data input/output constructs, users are currently free to attach any ontology concept to any web service. That is, the system does not provide any semantic verification regarding the inputs/outputs of the web services, thus ensuring that a web service requiring a specific type of input is actually bound to one.

Since the purpose of the automatic WSC module is to help the non-expert, but familiar with WSC concepts, user to create composite web services, our aim was to implement the manual WSC module with the same principal in mind. For this reason, users can opt to be informed about the intended use of each construct available for the manual workflow creation, and the created graphical workflows are validated against pre-defined rules whenever the users save them.

Some of the pre-defined rules were maintained from the original application. An example of this is that the ⟨*End*⟩ construct, necessarily being the last one in a workflow, must have at least one incoming sequence flow from another construct. Other rules were added in order to help the user to export a valid composite web service, e.g., that an ⟨*If-Then-Else*⟩ construct is required to have an outgoing ⟨*If*⟩ sequence flow and can optionally have an ⟨*Else*⟩ sequence flow. The full list of added rules is presented in Table I.

TABLE I. OWL-S RULES REGARDING THE MANUAL WSC MODULE.

#	**Description:**
1	A workflow with a ⟨*Split+Join*⟩ control construct must also contain an ⟨*End Split+Join*⟩ one and vice versa
2	A workflow with a ⟨*Repeat-While*⟩ control construct must also contain an ⟨*End Repeat-While*⟩ one and vice versa
3	Any Data construct that is inserted must have its binding set
4	Any Web Service Task construct that is inserted must have its binding set
5	A Data Input cannot have any input connectors
6	A Data Input must have an outgoing connector
7	A Data Output can only be present following a Task or an ⟨*If-Then-Else*⟩ gateway if the gateway has a Task as its source
8	A Data Output must have an incoming connector
9	An ⟨*If*⟩ or ⟨*Else*⟩ Sequence Flow can only have an ⟨*If-Then-Else*⟩ gateway as its source
10	A Start Event cannot have incoming connectors except if they originate from a Data Input
11	Only Web Services'Tasks and their Data Outputs can be contained between a ⟨*Split+Join*⟩ control construct and an ⟨*End Split+Join*⟩ one
12	An ⟨*If-Then-Else*⟩ gateway must have exactly one outgoing ⟨*If*⟩ Sequence Flow and an optional ⟨*Else*⟩ Sequence Flow

IV. EVALUATION FRAMEWORK

This section provides details regarding three use case scenarios that can be used to evaluate WSC systems. It presents the graphical workflows that correspond to these scenarios through the use of manual WSC module. Finally, **MADSWAN** is compared to two existing planning approaches, in order to evaluate its efficiency and effectiveness.

A. Use Case Scenarios

As mentioned in Section II.B, no standard web service test bed or test collection exists. For this reason, it is currently very hard to evaluate a WSC approach objectively against another one, which is a detriment to the ongoing research regarding efficient WSC composition approaches.

One of our goals was to create an evaluation framework that could be used and reproduced by other systems, that is, clearly define detailed use case scenarios that are based on an existing, open test collection. We decided for the use of OWL-S TC, since in the past few years it has been used extensively, as a test set in the recent S3 contests [74], or in several approaches in the recent literature [25, 39, 75].

We have designed three use cases, each based on the service descriptions contained in a single domain of OWL-S TC, although several minor modifications were made to the descriptions of some services, and a few descriptions were added to some domains in order to design more useful scenarios. All the modifications to the original test collection, as well as a full description of the use cases, can be found in [76]. Next, we describe the use case scenarios in detail, with the ontologies' concepts used in each one shown in parentheses (following the format "*ontology#concept*").

Each use case scenario has an increasing amount of non-determinism and complexity compared to the previous one.

1) Movie Database Use Case Scenario: The first use case (**MADSWAN**-UseCase$_1$ - MS-UC$_1$) is fully deterministic, allowing for the output of a fully serialized composite web service; it refers to a user who knows the title of a film (*my_ontology.owl#Film*) and wants to retrieve all the comedy films (*my_ontology.owl#ComedyFilm*) that exist with a similar title, along with their respective prices. That is, he desires to know all the relevant pricing information in regard to a comedy film, i.e., its regular price (*concept.owl#Price*), its maximum price (concept.owl# MaxPrice), and its price with (*concept.owl#TaxedPrice*) and without taxes (concept.*owl#TaxFreePrice*).

Finally, the returned comedy films along with their pricing information results should be stored in a database (*ontosem.owl#databas*e), so that he can remember to buy them in the future. MS-UC$_1$ uses the web services in the "Communication" domain of the test collection, with the relevant ones in regard to the use case amounting to a total of 58 semantic web services.

The rest of the scenarios incorporate non-deterministic elements, such as alternative outcomes in the output composite web service based on the availability of items, or user preferences between different types of products.

2) Online Bookstore Use Case Scenario: The second use case (MS-UC$_2$) refers to a client of a specific online bookstore who wants to purchase a book; the client can use three different methods to buy the book, with alternative outcomes being outputted by the composite web service based on whether the book is in stock at the online bookstore or not.

In detail, the scenario describes a situation in which an online bookstore's client initially provides as input to the composite WS a book title (*books.owl#Title*) or its ISBN (*portal.owl#ISBN*), his address (*order.owl#Address*) and a preferred method of payment for the purchased item. The available choices for the customer to pay for the selected book are using his credit (*finance_th_web.owl#credit_card*), cheque (*finance_th_web.owl#cheque_card*), or debit (*finance _th_web.owl#debit_card*) cards. In order to suit the purposes of this test case, we slightly altered the *finance_th_web* ontology and added the debit and cheque cards as subclasses of cash, and the credit card concept as a subclass of credit.

As a result, if the book is in stock at the specific e-bookstore, the final composite WS should use the specified method of payment to purchase the item and record the address for the item in the user's shopping cart to be shipped to. The result should be a purchased item (*order.owl# PurchasedItems*), and the output of information regarding the purchased book; specifically, the book's author (*books.owl#Author*), along with its type (hard-cover or paperback) (*books.owl#Book-Type*), and size (small, medium or large) (*books.owl#Size*). However, if the book is not in stock, the client should not be charged with a fee and no information regarding the item should be displayed to him. MS-UC$_2$ uses the web services in the "Education" domain of OWL-S TC, with the relevant ones being 285 in total.

3) Camera Search Use Case Scenario: The final scenario (MS-UC$_3$) also concerns the purchase of an item; the main difference with MS-UC$_2$ is that multiple sellers can be considered for this scenario and, as such, the composite web service may need to check with all of them to determine the availability of the item. Moreover, MS-UC$_2$ and MS-UC$_3$ differ in that the latter also incorporates the user's preferences in the scenario. The user is assumed to have a preference, that is, soft goal, towards an analog SLR camera model; however, he may settle for other cameras if the preferred one is not available from any seller.

In specific, MS-UC$_3$ refers to a user who initially provides as input to the composite web service the type of the camera he desires to buy, which is an analog, non-APS, standard SLR camera (*extendedCamera.owl#SLR*). However, if this camera is not available, the scenario presumes that the user would also be satisfied with buying another type of analog camera that has less features, specifically an analog, non-APS, standard compact camera (*extendedCamera.owl#Compact*). In either case, the user also provides the desired camera's product code (*extended Camera.owl#ProductCode*). Finally, though, if no store is found having any of the two desired cameras in stock, then he will settle for any kind of camera at all, whether analog or digital (*extendedCamera. owl#Camera*).

The user is also expected to state in advance which stores should be considered as alternatives for him to buy the camera from; in detail, he can choose to provide as input a shopping mall (*Mid-level-ontology.owl#ShoppingMall*), a retail store (*Mid-level-ontology.owl#RetailStore*) or a specific chain of retail stores, with the available ones being Walmart (*Mid-level-ontology.owl#WalmartStore*) and Media Markt (*Mid-level-ontology.owl#MediaMarktStore).* Finally, he can also state that all mercantile organizations can be considered as alternatives (*SUMO.owl#Mercantile Organization*). Again, the user can provide one or more inputs, but in this case all the available choices are considered equally preferable alternatives. We assume that the user does not differentiate between the alternative stores he has provided as input, as long as he finds the camera he desires in stock.

Having entered the desired product type along with its product code and the alternative stores that can be used to buy it from, the composite WS should find a store that sells this product and check whether it has it in stock or not. If it is in stock, it should add it to the user's shopping cart (*ShoppingCart.owl#ShoppingCartRequestItems*); if not, it should continue to search for another store that sells it. If it cannot find any store that has the SLR camera in stock, it will repeat the aforementioned process, this time searching for a compact camera. If no compact camera is in stock either, then the composite WS will search for any camera available in stock. The output of the service can only be an addition to the user's shopping cart or no action at all. MS-UC$_3$ makes use of the test collection's "Economy" domain and of a total of 359 semantic web services.

We validate the correctness of the automatic WSC solution plan for a problem by checking that all of the ontological concepts that were present in the hard goals of its goal state have been generated by the web services in the plan. That is, the initial state concepts, along with the outputs of the web services that take part in the plan should be a superset of the hard goals set by the user.

This set of scenarios provides use cases that can efficiently evaluate the capabilities of WSC methodologies in a way that is both reproducible and extensible. They allow for a system to showcase that it can indeed cope with non-determinism in the WSC domain, and output both sequential and conditional plans, with and without taking into account the user's preferences.

B. Experimental Results

In order to test the correctness and efficiency of the current version of the automatic WSC module, we empirically compared a preliminary version of the algorithm presented in Section III.B against two existing planning

TABLE II. COMPARISON OF PLANNING TIMES FOR POND, LPG-TD AND MADSWAN.

	Opt	POND-EHC		POND-A*		POND-AO*		LPG-td		MadSwan	
		Plan Length	*Time*	*Plan Length*	*Time*	*Plan Length*	*Time*	*Average Plan Length*	*Average Time*	*Plan Length*	*Time*
MS-UC$_1$	6	6	**0.0024**	6	0.0068	6	0.0032	6.04	0.0040	6	0.0169
P-10	6	6	0.0016	6	0.0020	6	**0.0016**	6	0.0066	6	0.0172
P-100	6	6	0.0036	6	0.0028	6	**0.0026**	6	0.0124	6	0.0117
P-1000	5	5	0.2640	5	2.5404	5	0.2742	5.76	0.1256	5	**0.0996**

a. The time measurements are in seconds; for the stochastic planner, LPG-td, they represent the median values of 100 runs. "Opt" represents the optimal length of each problem.

approaches; the first is a WSC system, PORSCE II [26], and the second a state-of-the-art contingent planner, POND [77].

We used two domains, one taken from the evaluation of [26], in order to provide direct comparison with PORSCE II, and one corresponding to MS-UC$_1$. The experiments were run on a PC using a Dual-Core Intel i5 processor running at 1.6GHz and allowing at most 4GB memory.

The domain taken from the PORSCE II system is deterministic in nature and, similarly to MS-UC$_2$, it models a user of a bookstore who wants to purchase a book. It is however a simplified version of MS-UC$_2$, as it does not feature non-determinism or any choices on behalf of the user. He is presumed to have a credit card that he can use to purchase a book, and the book is always considered to be in stock.

As in [26], we test three different versions of this domain, named *P-x*, "*P*" symbolizing the PORSCE system and *x* representing the number of web services participating in the problem. Since the web services are translated to PDDL actions, each version is increasingly more complex than the previous ones, the first consisting of 10 web services (*P-10*), the second of 100 (*P-100*) and the last of 1000 ones (*P-1000*). In the first two versions, the optimal plan length, i.e., the least amount of web services needed to achieve the desired goal, is 6, whereas the last version has an optimal plan length of 5. All of the aforementioned planning domains and problems are available at [78].

The second domain in the test set is from our own framework; since the proposed contingent algorithm has not yet been integrated in the online prototype, the use case scenario tested is the deterministic one, MS-UC$_1$. Specifically it is a relatively simpler version of it, with 21 web services taking part in the. The optimal plan length is also 6.

POND is a planner able to solve partially observable and/or non-deterministic problems by searching forward in the space of belief states, guided by a relaxed plan heuristic. It generates conformant and conditional plans using various search algorithms, specifically A*, AO* [79], LAO* [80], and Enforced Hill-Climbing (EHC) [81]. In our experimental setup we executed three versions of POND version 2.2, using the A*, AO* and EHC search algorithms.

As aforementioned in Section II, PORSCE II relies on two alternative planning systems, JPlan and LPG-td. For our experiments, we compare **MadSwan** against LPG-td, as the authors of [26] concluded that its performance was by far superior to that of JPlan. LPG (Local search for Planning Graphs) is a sub-optimal anytime planner based on stochastic local search and planning graphs; its search space comprises action graphs, that is, particular subgraphs of the planning graph representing partial plans.

The experimental results are presented in Table II. In all cases, the time reported is in seconds and represents the total planner time needed to reach a solution. The planner with the best performance in each problem is highlighted in bold, and the optimal plan length (*Opt*) is shown next to each problem.

The experiments indicate that the number of web services available for WSC is a crucial factor in regard to the planner's efficiency, even if not all services are necessarily useful for the achievement of the goal. The results in Table II indicate that, generally, as the number of web services participating in each problem increases, so does the time required to solve it, although not linearly. This assumption is corroborated by the results of the comparison between MS-UC$_1$ and *P-10*, the two problems that comprise of just a few web services. MS-UC$_1$ requires more time than *P-10* for all versions of POND, and almost the same time for **MadSwan,** but a little less time for LPG-td. This is not surprising, as the two problems share a common optimal plan length and MS-UC$_1$ comprises almost double the web services than P-10.

Moreover, although all the other problems in the experiments have a larger optimal length, *P-1000* appears to be by far the most difficult problem in the test set. The increase in the required time to solve *P-1000* for POND- A* is almost a thousand times more than for the second most difficult problem, *P-100*, a hundred times more for POND-EHC and POND-AO*, and tenfold for LPG-td and **MadSwan**.

Although the number of web services comprising a particular problem is important, though, its difficulty is not dependent solely on this factor; e.g., **MadSwan** manages to solve *P-100* faster than it does *P-10*. This fact can be mainly attributed to the complexity of the problem itself. The preprocessing phases of both LPG-td and all the versions of POND search for useless actions in the domain and allow the planners to ignore them during search (or simply prune them at parsing time). For this reason, both planners report that *P-10* consists of 10 (relevant) actions, whereas *P-100* of just 6.

Preprocessing leads to simpler domains, that is, for POND (that includes such facts in its output), *P-10* comprises 15 state variables while *P-100* comprises only 6. As such, both planners spend less time actually searching for a valid plan; on the other hand, though, this means that for relatively easy domains such as *P-100*, the vast majority of the total planner time is spent in the preprocessing phase. **MadSwan** is more straightforward, devoting less time to preprocessing techniques; most of its total time is spent on search, allowing it to solve the – easier – *P-100* problem quicker than *P-10*, despite the fact that it searches among more web services/actions.

In general, the problems seem to be almost trivial for all planners, with the exception of *P-1000* for POND- A*, which has a significantly worse performance for this problem, both compared to the other planners and to its performance in the rest of the problems. All versions of POND, as well as **MadSwan,** find solutions of optimal length for all problems. LPG-td, however, being a stochastic anytime non-optimal planner, returns plans of slightly worse median length for MS-UC$_1$ and (mainly) *P-1000*. **MadSwan** needs more time than all versions of POND for the smaller problems (MS-UC$_1$, *P-10,* and *P-100*), as well as than LPG-td for the two smallest ones. On the other hand, it is faster than all planners for the most complex problem, *P-1000*.

It is important to note that since POND is not a WSC approach, but a standard PDDL planner, we experimentally tested only the efficiency of the planners on translated WSC domains to planning ones, and not on the whole process that **MadSwan** typically follows. However, a major bottleneck of the solution of WSC problems seems to be the translation process from WSC domains to planning ones and vice versa.

Table III presents the results of our experiments regarding the average transformation time per web service description. In [26], it is reported that the average transformation time per web service converged to approximately 0.8 seconds. In our experiments, though, the necessary time was considerably less, converging to approximately 0.1 seconds per web service when translating a web service registry analogous to the size of the entire OWL-S TC.

Moreover, our experiments show a decrease in the average time required per web service translation as the total number of web services in the set increases, in contrast to the results reported in [26]. Since our translation process is based on the one in [26], we can assume that this fact, along with the significant improvement in the average transformation time per web service, can mainly be attributed to our different hardware setup, as well as minor optimizations in the original source code, mainly concerning the used data structures.

TABLE III. COMPARISON OF TRANSLATION TIMES PER WEB SERVICE FOR PORSCE II AND MADSWAN.

WSs	PORSCE II	MadSwan
	Time	
10	0.4590	0.3974
21	-	0.2836
100	0.7000	0.1774
1000	0.7920	0.1110

a. All time measurements are in seconds and represent the median values of 100 runs. All the data for PORSCE II have been taken from [25].

The results of Table III indicate that most of the required computational effort for the automatic WSC process is attributed to the translation process of the original domain to the planning one, and not to the problem's solution itself. This is apparent by the fact that even when the average transformation time per web service converges to its lowest value (0.1110 seconds), the transformation time of a single web service is larger than the solution of a problem containing the same number of web services/actions (0.0996 seconds); that is, the time required by the planner to find a solution is less than one hundredth of the total time needed for the entire WSC process.

C. *Use Case Scenarios in the Manual Web Service Composition Module*

The existing implementation of the manual composition module is capable of producing graphical workflows that correspond to the three use case scenarios that were presented in Section IV-A. Fig. 7, 8 and 9 present the manually created workflows representing MS-UC$_1$, MS-UC$_2$ and MS-UC$_3$, respectively. In all figures, the labels of the connecting sequence flows have been removed so as to ease their legibility. The files used by the system's internal load and save functionalities for the three use case scenarios can be found at [82].

Using the automatic WSC module, the solution plan for the tests cases, e.g., MS-UC$_1$, requires just a fraction of a second; it is evident that even an experienced user would require at least a few minutes in order to accomplish a similar result visually in the manual WSC module. For that reason, the manual WSC module is mainly useful for the visualization of WSC workflows, and also for the creation of a basic "sketch" of a composite web service description, which can then be manually completed by experts.

We are currently working on the transformation of the created graphical workflows to their respective OWL-S descriptions and vice versa. The current functionalities of the manual WSC module are showcased in [65].

V. Conclusion and Future Work

In this article, we presented our work regarding the implementation of an online WSC system, named **MadSwan**, which makes use of AI planning techniques and of already freely available web service-related components. We provide a link to a demo of **MadSwan**, albeit still in alpha version, which showcases the system's functionalities.

This publicly available prototype is, to the best of our knowledge, the first online application of its kind able to support various stages of the WSC process. The fact that the presented system allows its users to store/retrieve web services to/from a registry, edit the ones already stored in it, and create new workflows both manually and automatically, all through an online interface, constitute a unique set of functionalities for such a system.

The current WSC approaches face several limitations, the most important of which is the plethora of available standards, both in relation to the semantic description of services and their underlying implementation. We acknowledge these limitations, which require substantial standardization efforts and coordination between the various service providers. For this reason, **MadSwan** is based on open source components, such as the manual WSC composition module and the registry, and utilizes existing datasets, i.e., OWL-S TC, and the current web service standards. This approach fortifies the web service principles the system is based on, and allows a more efficient comparison of the system to similar ones. Thus, we hope that our efforts constitute a step toward overcoming such incompatibility obstacles.

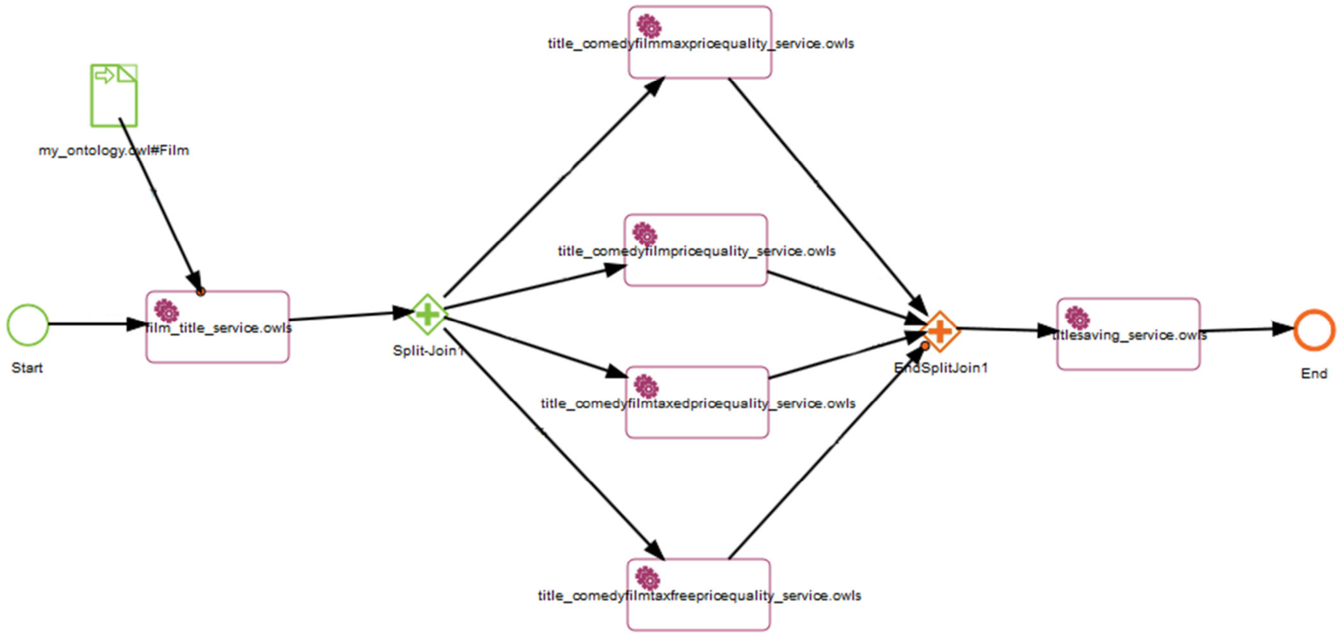

title_comedyfilmmaxpricequality_service.owls
my_ontology.owl#Film
title_comedyfilmpricequality_service.owls
Start
film_title_service.owls
Split-Join1
EndSplitJoin1
titlesaving_service.owls
End
title_comedyfilmtaxedpricequality_service.owls
title_comedyfilmtaxfreepricequality_service.owls

Figure 7. Movie database scenario workflow.

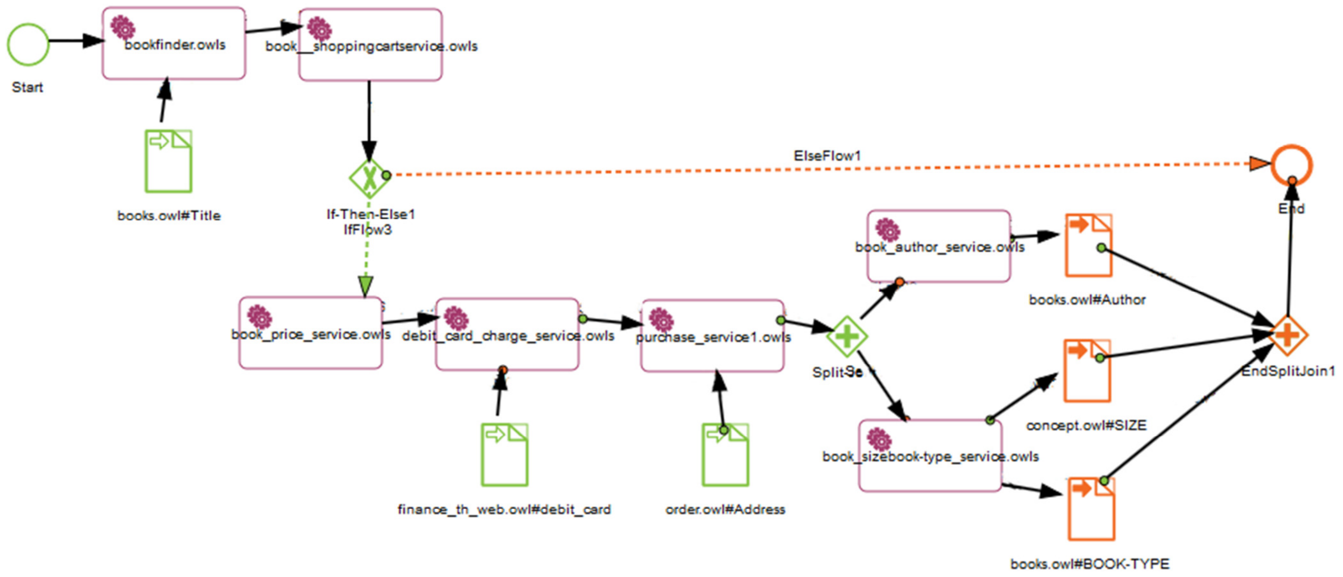


Figure 8. Online bookstore scenario workflow.

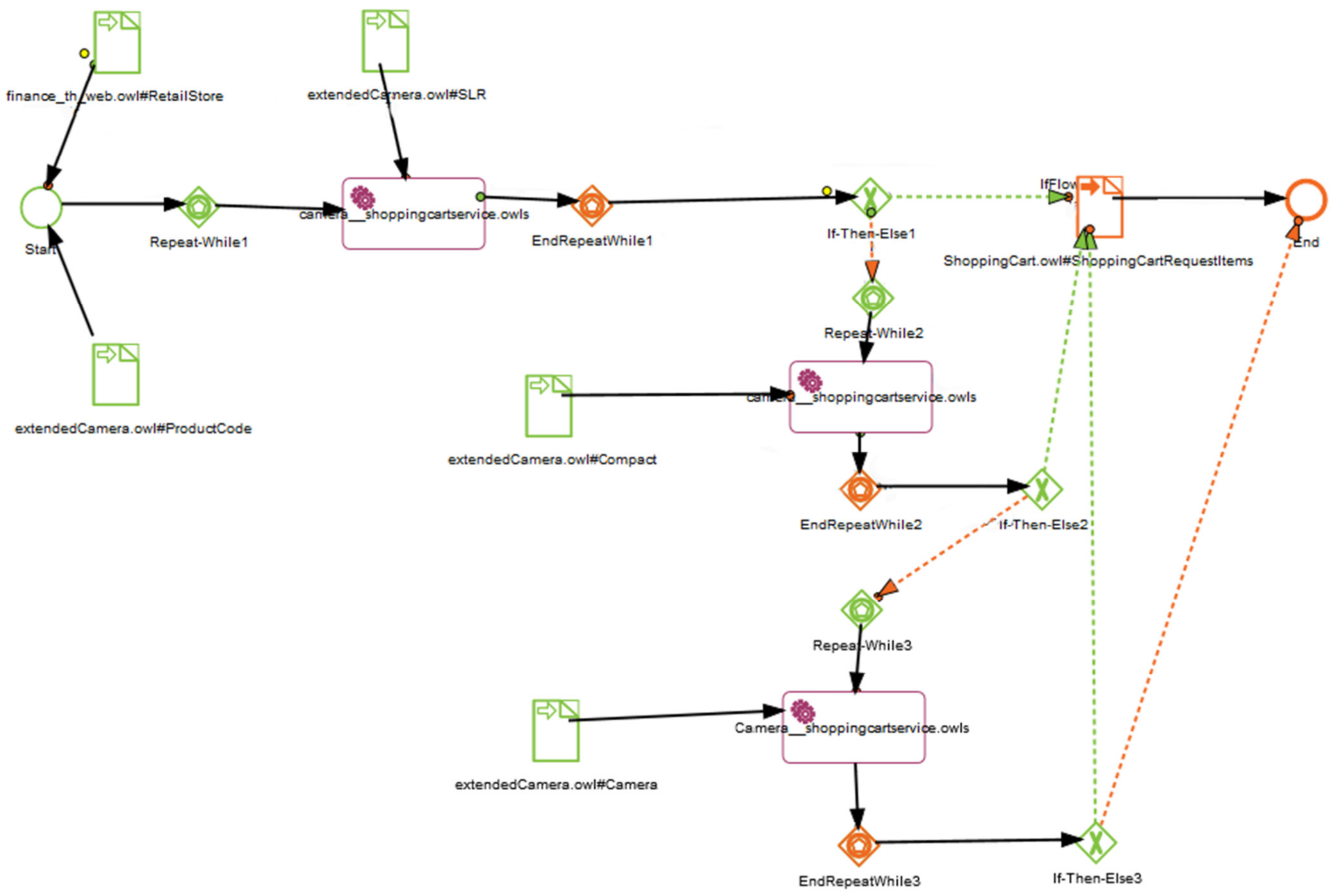

finance_th_web.owl#RetailStore
extendedCamera.owl#SLR
Start
Repeat-While1
camera__shoppingcartservice.owls
EndRepeatWhile1
If-Then-Else1
IfFlow
ShoppingCart.owl#ShoppingCartRequestItems
End
Repeat-While2
extendedCamera.owl#ProductCode
camera__shoppingcartservice.owls
extendedCamera.owl#Compact
EndRepeatWhile2
If-Then-Else2
Repeat-While3
Camera__shoppingcartservice.owls
extendedCamera.owl#Camera
EndRepeatWhile3
If-Then-Else3

Figure 9. Camera search scenario workflow.

Currently, we are working on the design and implementation of a non-deterministic planning module, with the intention of finding plans that withstand various contingencies that can occur in non-deterministic domains. As demonstrated by our experiments, the current version of the prototype is able to find in a reasonable amount of time plans that solve deterministic WSC problems. The solutions are translated back to web service description files to be saved back to the registry for future use.

The article also presents in detail three use case scenarios that can be used to evaluate WSC processes. We established experimentally that the current, deterministic, version of the algorithm used in **MadSwan** is competitive with both PORSCE II and POND in a variety of test cases. Furthermore, the efficiency of the translation process of the original WSC to a planning one was shown to be adequately fast for even a large repository of web services, e.g., one that contains all the web services in OWL-S TC.

Additionally, we provided implementation details in regard to the manual WSC module. This module can be used as an alternative counterpart to the automatic one; our goal, however, is that it will be used as a standalone application in order to help users create composite web service descriptions more efficiently compared to actually writing the entire description files by hand. Such a tool would allow users to familiarize with the concepts of web services and OWL-S, or simply to visualize composite OWL-S description files.

For the future, we plan to incorporate the contingent planning algorithm in the online prototype, and acquire experimental results for non-deterministic WS domains. We are also working on the semantic verification of inputs/outputs for the manual WSC module. Moreover, one of our long-term goals is to allow the execution of semantic web services, through the use of external tools such as SPEX (SPecification and EXecution tool) [83]. Finally, we aim to support automatic translation of arbitrary OWL-S composite web service description into the graphical workflows of the manual component.

Acknowledgments

This research has been co-financed by the European Union (European Social Fund – ESF) and Greek national funds through the Operational Program “Education and Lifelong Learning” of the National Strategic Reference Framework (NSRF) - Research Funding Program: Heracleitus II. Investing in knowledge society through the European Social Fund.

The authors would like to thank Graham Crosmarie of the PetalsLink Research & Development Team for his time and advice in regard to Petals BPM, as well as the anonymous reviewers for their valuable comments.

REFERENCES

[1] G. Markou and I. Refanidis, "Towards Automatic Non-Deterministic Web Service Composition", Proc. *7th International Conference on Internet and Web Applications and Services* (ICIW'12), May 2012.

[2] T. Berners-Lee, J. Hendler, and O. Lassila, "The Semantic Web", *Sci. Am.*, May 2001, pp. 29-37.

[3] S. Narayanan and S.A. McIlraith, "Simulation, verification and automated composition of web services", Proc. *11th International World Wide Web Conference* (WWW'02), May 2002, pp. 77-88.

[4] D. Martin, M. Burstein, J. Hobbs, et al. 2004. OWL-S: Semantic Markup for Web Services, http://www.w3.org/Submission/OWL-S/ [Dec. 12, 2013]

[5] A. Muscholl and I. Walukiewicz, "A Lower Bound on Web Services Composition", *Log. Meth. Comput. Sci.,* vol. 4 (2), 2008.

[6] W. Nam, H. Kil, and D. Lee, "On the Computational Complexity of Behavioral Description-Based Web Service Composition", *Theor. Comput. Sci.*, vol. 412 (48), 2011, pp. 6736-6749.

[7] J.Rao and X. Su, "A Survey of Automated Web Service Composition Methods", Proc. *1st international conference on Semantic Web Services and Web Process Composition* (SWSWPC '04), July 2004.

[8] S. J. Russell, and P. Norvig, "Planning", in *Artificial Intelligence: A Modern Approach*, 2nd ed., Upper Saddle River, NJ, Prentice Hall, 2003, ch. 11, pp. 375–459.

[9] M. Ghallab, D. S.Nau, and P. Traverso, *Automated Planning: Theory and Practice*, 1st ed., San Francisco, CA, Morgan Kaufmann, 2004.

[10] L. Pryor and G. Collins, "Planning for contingencies: A decision-based approach", *J. Artif. Intell. Res*, vol. 4, 1996, pp. 287-339.

[11] H. L. S. Younes and M. L. Littman, "*PPDDL1.0: An extension to PDDL for expressing planning domains with probabilistic effects*", Technical Report. School of Computer Science, Carnegie Mellon University, 2004.

[12] G. Markou and I. Refanidis, "Towards an Automatic Non-Deterministic Web Service Composition Platform", Proc. *8th International Conference on Next Generation Web Services Practices* (NWeSP '12), Nov. 2012.

[13] Q. Z. Sheng, B. Benatallah, Z. Maamar, and A. H. H. Ngu, "Configurable Composition and Adaptive Provisioning of Web Services", *IEEE Trans. Services Comput*., vol. 2(1), 2009, pp. 34-49.

[14] T. Bray, J.Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau, "Extensible Markup Language (XML) 1.0", 2008. http://www.w3.org/TR/REC-xml/ [Dec. 12, 2013]

[15] UDDI version 3.0.2, http://uddi.org/pubs/uddi_v3.htm [Dec. 12, 2013]

[16] J. Camara, J. A. Martin, G. Salaun, et al., ITACA: An integrated toolbox for the automatic composition and adaptation of Web services. Proc. *31st International Conference on Software Engineering* (ICSE), May 2009, pp. 627-630.

[17] A. Alves,A. Arkin, S. Askary, et al., "Web Services Business Process Execution Language Version 2.0", 2007. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html [Dec. 08, 2013]

[18] E. Christensen, F. Curbera, G. Meredith, and S.Weerawarana, "Web Services Description Language (WSDL) 1.1", 2001. http://www.w3.org/TR/wsdl [Dec. 12, 2013]

[19] Microsft Corporation, "Windows Workflow Foundation", 2007. http://msdn.microsoft.com/en-us/library/vstudio/ms735 967(v=vs.90).aspx [Nov. 23, 2013]

[20] J. Cubo and E. Pimentel, DAMASCo: A Framework for the Automatic Composition of Component-Based and Service-Oriented Architectures. *The 5th European Conference on Software Architecture (ECSA),* Sept. 2011, pp. 388-404.

[21] E. Sirin, B. Parsia, D. Wu, J. Hendler, and D. Nau, "HTN planning for web service composition using SHOP2", *J. Web Semant.,* vol. 1, no. 4, Oct. 2004, pp. 377-396.

[22] M. Fox and D. Long, "PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains". *J. Artif. Intell. Res*, vol. 20, Dec. 2003, pp. 61–124.

[23] J. Hoffmann, P. Bertoli, M. Helmert, and M. Pistore, "Message-based web service composition, integrity constraints, and planning under uncertainty: a new connection", *J. Artif. Intell. Res*, vol. 35, May 2009, pp.49-117.

[24] J. Hoffmann and R. Brafman, "Conformant Planning via Heuristic Forward Search: A New Approach", *Artif. Intell.*, vol. 170 (6-7), 2006, pp. 507 – 541.

[25] M. Klusch, A. Gerber, and M. Schmidt, "Semantic web service composition planning with OWLS-Xplan", Proc. *1st International AAAI Fall Symposium on Agents and the Semantic Web*, Nov. 2005.

[26] O. Hatzi, D. Vrakas, M. Nikolaidou, et al., "An integrated approach to automated semantic web service composition through planning", *IEEE Trans. Service Computing*, April 2011, pp. 301-308.

[27] JPlan: Java GraphPlan Implementation http://sourceforge.net/ projects/jplan/ [Nov. 18, 2013]

[28] A. Gerevini, A. Saetti, and I. Serina, "An approach to temporal planning and scheduling in domains with predictable exogenous events", *J. Artif. Intell. Res*, vol. 25, Jan. 2006, pp.187-231.

[29] M. Kuzu and N. Cicekli, "Dynamic planning approach to automated web service composition", *J. Appl. Intell.*, vol. 36, January 2012, pp.1-28.

[30] H.S. Kim and I.C. Kim, "Mapping semantic web service descriptions to planning domain knowledge", Proc. *International Federation for Medical and Biological Engineering* (IFMBE), Aug. 2006.

[31] E. Onaindia, O. Sapena, L. Sebastia, and E. Marzal, "SimPlanner: An Execution-Monitoring System for Replanning in Dynamic Worlds", Proc. *10th Portuguese Conference on Artificial Intelligence* (EPIA 2001), Dec. 2001, pp. 393-400.

[32] A. Macdonald, "*Service composition with hyper-programming*". Technical Report, University of St Andrews, 2007.http://www.cs.st-andrews.ac.uk/~angus/docs/yawsa/ final_report.pdf [Dec. 12, 2013]

[33] X. Du, W. Song, and M. Munro, "Using common process patterns for semantic web services composition", Proc. *15th International Conference on Information Systems Development* (ISD'06), Sept. 2006.

[34] SUPER - Semantics Utilized for Process management within and between EnteRprises, http://www.ip-super.org/ 08/30/2013

[35] Business Process Model and Notation (BPMN) Version 2.0, http://www.omg.org/spec/BPMN/2.0/ [Dec. 07, 2013]

[36] D. Roman, U. Keller, H. Lausen, et al., "Web Service Modeling Ontology", *Applied Ontology*, 2005, pp. 77-106.

[37] M. Chan, J. Bishop, and L. Baresi, "*Survey and comparison of planning techniques for web services composition*". Technical Report. University of Pretoria, 2007. http://polelo.cs.up.ac.za/ papers/Chan-Bishop-Baresi.pdf [Dec. 12, 2013]

[38] R.W. Feenstra, M. Janssen, and R.W. Wagenaar, "Evaluating Web Service Composition Methods: The need for including Multi-Actor Elements", *Elect. J. of e-Gov. (EJEG)*, vol.5 (2), 2007, pp. 153-164.

[39] G. Canfora and M. Di Penta, "Testing services and service-centric systems: Challenges and opportunities", *IT Prof.*, vol. 8, 2006, pp. 10–17.

[40] M. Bozkurt, M. Harman, and Y. Hassoun, "Testing Web Services: A Survey", Technical Report, King's College London Centre for Research on Evolution, Search & Testing, TR-10-01, 2010.

[41] P. Bartalos and M. Bielikova, "Automatic Dynamic Web Service Composition: A Survey and Problem Formalization", Comput. Inform., vol. 30 (4), 2011, pp. 793-827.

[42] M. Alrifai, T. Risse, P.Dolog, and W. Nejdl, "A Scalable Approach for QoS-Based Web Service Selection", Proc. *1st International Workshop on Quality-of-Service Concerns in Service Oriented Architectures* (QoSCSOA '08), Dec. 2008, pp. 190-199.

[43] D.H. Shin and K.H. Lee, "An Automated Composition of Information Web Services based on Functional Semantics", Proc. *IEEE Workshop on Web Service Composition and Adaptation* (WSCA '07), July 2007.

[44] F. Rosenberg, M.B. Müller, P. Leitner, et al., "Metaheuristic Optimization of Large-Scale QoS-aware Service Compositions". Proc. *IEEE International Conference on Services Computing* (SCC '10), June 2010, pp. 97-104.

[45] W. Jiang, C. Zhang, Z. Huang, et al., "QSynth: A Tool for QoS-aware Automatic Service Composition", Proc. *8th IEEE International Conference on Web Services* (ICWS 2010), July 2010, pp. 42-49.

[46] Web Service Challenge 2009. http://ws-challenge.georgetown.edu/wsc09/ [Dec. 07, 2013]

[47] Y. Li, J. Huai, H. Sun, T. Deng, and H. Guo , "PASS: An Approach to Personalized Automated Service Composition", Proc. *IEEE International Conference on Services Computing* (SCC '08), July 2008, pp. 283-290.

[48] K. Chen, J. Xu, and S. Reiff-Marganiec, "Markov-HTN planning approach to enhance flexibility of automatic web service composition", Proc. *IEEE International Conference on Web Services* (ICWS'09), July 2009, pp. 9-16.

[49] Y. Bo and Q. Zheng, "A method of semantic web service composition based PDDL", Proc. *IEEE International Conference on Service-Oriented Computing and Applications* (SOCA '09), Dec. 2009, pp. 1-4.

[50] S. Kona, A. Bansal, M.B. Blake, and G. Gupta, "Generalized Semantics-Based Service Composition", Proc. *IEEE International Conference on Web Services (ICWS'08)*, Sept. 2008.

[51] SemWebCentral OWL-S Service Retrieval Test Collection, http://semwebcentral.org/frs/?group_id=89 [Dec. 01, 2013]

[52] M.B. Blake, W.K.W. Cheung, and A. Wombacher, "WSC-06: The Web Service Challenge", Proc. *8th IEEE International Conference on E-Commerce Technology and 3rd IEEE International Conference on Enterprise Computing, E-Commerce, and E-Services* (CEC/EEE'06), June 2006, pp. 422-423.

[53] J. McGovern, S.Tyagi,M. Stevens, and S. Mathew. *Java Web Services Architecture*. Morgan Kaufmann, 2003, pp. 35.

[54] N. Srinivasan, M. Paolucci, and K. Sycara, "Adding OWL-S to UDDI, implementation and throughput", Proc *1st International Workshop on Semantic Web Services and Web Process Composition* (SWSWPC '04), July 2004.

[55] J. Luo, B.E. Montrose, A. Kim, A. Khashnobish, and M. Kang, "Adding OWL-S support to the existing UDDI infrastructure". Proc. *IEEE International Conference on Web Services* (ICWS '06), Sept. 2006, pp. 153-162.

[56] C. Pedrinaci, D. Liu, M. Maleshkova, et al., "iServe: a linked services publishing platform", Proc. *Ontology Repositories and Editors for the Semantic Web Workshop at the 7th Extended Semantic Web Conference* (ORES '10), May 2010.

[57] J. Farrell and H. Lausen, "Semantic Annotations for WSDL and XML Schema", 2007. http://www.w3.org/TR/sawsdl/ [Nov. 23, 2013]

[58] D. Fensel, F. Fischer, J.Kopecký, et al., "WSMO-Lite: Lightweight Semantic Descriptions for Services on the Web", 2010. http://www.w3.org/Submission/2010/SUBM-WSMO-Lite-20100823/ [Dec. 09, 2013]

[59] D. Beckett, "RDF/XML Syntax Specification", 2004. http://www.w3.org/TR/rdf-syntax-grammar/ [Dec. 09, 2013]

[60] D. Beckett, T. Berners-Lee, E. Prud'hommeaux, and G.Carothers, "Turtle - Terse RDF Triple Language", 2013. http://www.w3.org/TR/turtle/ [Nov. 20, 2013]

[61] MADSWAN YouTube channel, http://goo.gl/kK5PAV [Dec. 12, 2013]

[62] P. Birchmeier, "*Semi-automated semantic web service composition planner supporting alternative plans synthesis and imprecise planning*". Diploma Thesis, University of Zurich, 2007. http://www.ifi.uzh.ch/pax/uploads/pdf/publication/1691/Birchmeier_Peter.pdf [Nov. 23, 2013]

[63] MADSWAN execution example files, http://ai.uom.gr/gmarkou/Files/IJAIT/ [Dec. 12, 2013]

[64] Sample semantic web service description files, http://ai.uom.gr/gmarkou/Files/OWL-S/ [Dec. 12, 2013]

[65] MADSWAN online prototype – alpha version, http://madswan.uom.gr/ [Dec. 10, 2013]

[66] J.Hoffmann and R.Brafman, "Contingent planning via heuristic forward search with implicit belief states". Proc *15th International Conference on Automated Planning and Scheduling* (ICAPS '05), June 2005, pp. 71-80.

[67] A. Blum and M. Furst, "Fast planning through planning graph analysis", *Artif. Intell.,*vol. 90, 1997, pp. 281–300.

[68] I. Little, "Paragraph: A graphplan based probabilistic planner", Proc. *5th International Planning Competition* (IPC-5), 2006.

[69] P. E. Hart, N. J. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths", *IEEE Trans. Syst. Sci., Cybern.,* vol. 4 (2), 1968, pp.100–107.

[70] B. Bonet and H. Geffner, "Planning as heuristic search", *Artif. Intell.*, vol. 129(1-2), 2001, pp. 5-33.

[71] E. Ziaka, D. Vrakas, and N. Bassiliades, "Translating web services composition plans to OWL-S descriptions", Proc. *3rd International Conference on Agents and Artificial Intelligence* (ICAART '11), Jan. 2011, pp. 167-176.

[72] OWL-S API introduction, http://on.cs.unibas.ch/owls-api/index.html [Dec. 06, 2013]

[73] Petals BPM architecture overview, https://research.petalslink.org/display/petalsbpm/Petals+BPM+-+Open+source+BPMN+2.0+modeler [Dec. 12, 2013]

[74] S3 Contest: Retrieval performance evaluation of matchmakers for semantic web services, http://www-ags.dfki.uni-sb.de/~klusch/s3/html/2011.html [Dec. 4, 2013]

[75] A. Mesmoudi, M. Mrissa, and M. Hacid, "combining configuration and query rewriting for web service composition", Proc. *IEEE International Conference on Web Services* (ICWS '11), July 2011, pp. 113-120.

[76] G. Markou, "*Heracleitus II - WSC Use Case Scenarios*". Technical report. University of Macedonia, 2012. Available at http://ai.uom.gr/gmarkou/Files/Mad_Swan_Use_Case_Scenarios.pdf [Dec. 12, 2013]

[77] D. Bryce, S. Kambhampati, and D.E. Smith, "Planning Graph Heuristics for Belief Space Search". *J. Artif. Intell. Res*, vol. 26, May 2006, pp. 35–99.

[78] Evaluation Domain and Problems, http://ai.uom.gr/gmarkou/Files/EvaluationDomains/ 12/012/2013

[79] N. Nilsson, *Principles of Artificial Intelligence*, San Francisco, CA, Morgan Kaufmann, 1980.

[80] E. Hansen and S. Zilberstein, "LAO: A heuristic-search algorithm that finds solutions with loops", *Artif. Intell.*, vol. 129 (1-2), 2001, pp. 35 – 62.

[81] J. Hoffmann and B. Nebel, "The FF planning system: Fast plan generation through heuristic search". *J. Artif. Intell. Res*, vol. 14, Jan. 2001, pp. 253–302.

[82] Use case scenarios' output files screenshots, http://ai.uom.gr/gmarkou/Files/IJAIT/Scenarios(BPMNandScreenshots).rar [Dec. 12, 2013]

[83] J.T.E. Timm and G.C. Gannod, "Grounding and Execution of OWL-S Based Semantic Web Services", Proc. *IEEE International Conference on Services Computing* (SCC '08), July 2008, pp. 588-592.