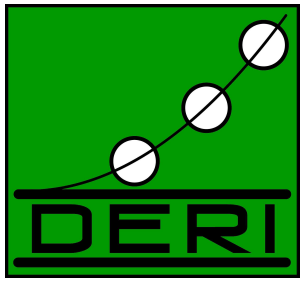


SEMANTIC DISCOVERY CACHING: PROTOTYPE & USE CASE EVALUATION

Michael Stollberg and Martin Hepp

DERI TECHNICAL REPORT 2007-04-07

APRIL 2007

DERI Galway
University Road
Galway, Ireland
www.deri.ie

DERI Innsbruck
Technikerstrasse 21a
Innsbruck, Austria
www.deri.at

DERI Korea
Yeonggun-Dong, Chongno-Gu
Seoul, Korea
korea.deri.org

DERI Stanford
Serra Mall
Stanford, USA
www.deri.us

DERI TECHNICAL REPORT
DERI TECHNICAL REPORT 2007-04-07, APRIL 2007

SEMANTIC DISCOVERY CACHING (SDC): PROTOTYPE & USE CASE EVALUATION

Michael Stollberg and Martin Hepp[1]

Abstract. This technical report presents the prototype implementation of *Semantic Discovery Caching* (short: SDC), a technique for efficient and scalable discovery for Semantic Web services. It captures the results of design time discovery (usability of Web services for generic goal descriptions), and utilizes this knowledge for efficient runtime discovery (find a usable Web service for a concrete client request) by pre-filtering and minimizing the number of matchmaking operations. The SDC technique is a realization of the Web Service discovery framework envisioned for the Web Service Modelling Ontology WSMO. It is based on a profound, formalized specification of its elements and operations in first-order logic (FOL). While the theoretical and formal foundations have been presented in other works, this document presents the prototype implementation and an evaluation with respect to the achievable increase in efficiency and scalability within the shipment scenario defined in the SWS Challenge.

Keywords: Semantic Web Services, Goals, Discovery, Efficiency, Scalability

[1]Digital Enterprise Research Institute (DERI) Innsbruck, University of Innsbruck, Technikerstraße 21a, A-6020 Innsbruck, Austria. eMail: {michael.stollberg,martin.hepp }@deri.org.

Acknowledgements: This material is based upon works supported by the EU commission and by the Austrian government.

The authors dedicate special thanks to Holger Lausen, Mick Kerrigan, Thomas Haselwanter, and Adina Sirbu for help during the implementation.

Contents

List of Figures

List of Tables

1 Introduction

This report provides a documentation of the prototype of Semantic Discovery Caching (short: SDC), an efficient and scalable Web service discovery engine, along with an evaluation in a real world scenario.

SDC is an optimization technique for the Web service discovery process that addresses the challenge of *efficiency* (the required time for finding a usable Web service) and *scalability* (the ability to deal with large numbers of available Web services). These are critical success factors for the adaptation of Semantic Web service technologies in real world scenarios. The SDC technique is a novel approach that adopts the concept of caching to the context of Web service discovery. It provides one possible realization of the discovery framework envisioned within the Web Service Modeling Ontology WSMO [3] (see homepage: `www.wsmo.org`), with a primary focus on functional aspects.

Figure 1 gives an overview of the approach as a data flow diagram. At design time, Web services for **goal templates** are discovered by matchmaking of their formal functional descriptions. A goal template is a generic client objective description; the usability of each Web service for a goal template is expressed in terms of matching degrees (exact, plugin, subsume, intersect, disjoint). The result is cached in the so-called **SDC-Graph** that organizes goal templates with respect to their semantic similarity, and captures the minimal knowledge of usable Web services. At runtime, a concrete client request is formulated in terms of a **goal instance** that instantiates a goal template with concrete inputs; this is referred to as the *goal formulation* process. The runtime discovery finds one usable Web service. This is optimized by using the knowledge kept in the SDC Graph, in particular: (1) only those Web service usable for the corresponding goal template are potential candidates for the goal instance (**pre-filtering**), (2) for Web services that are usable for a goal template under the degrees exact or plugin no matchmaking is necessary for discovery at runtime (**minimizing the number of necessary matchmaking operations**).

The SDC technique is based on a profound formalization. While this document focusses on the prototype implementation and the evaluation in a use case scenario, we refer to the following documents for the theoretical and formal foundations: [8] presents the formal foundations for the two-phase Web service discovery, including the differentiation of goal templates and goal instances, the definition and formal semantics of functional descriptions for Web services and goals, and the formal definition of the design time and runtime Web service discovery operations; [7] provides the theoretical foundations of the SDC technology, including the definition of the SDC Graph, and the specification of all operations required for SDC-enabled Web service discovery as illustrated in Figure 1.

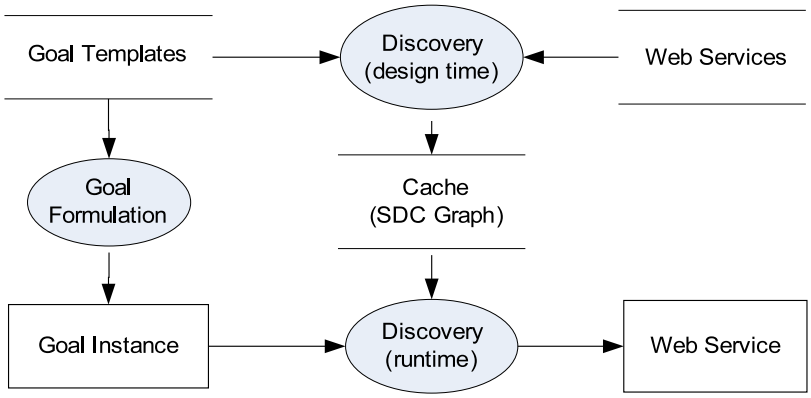


Figure 1: Overview of the SDC-Enabled Web Service Discovery

This document is organized as follows. Section 2 provides a documentation of the SDC prototype. This is realized as a component of the Web Service Execution Environment WSMX (see `www.wsmx.org`), the reference implementation of WSMO. The prototype is open source software, available for download from the SDC homepage at `http://members.deri.at/~michaels/software/sdc/`.

The main result of this work is the achieved increase in efficiency and scalability of the discovery process. For demonstration, we have compared the SDC runtime discovery with an engine that applies the same matchmaking techniques but does not make use of the cached design time discovery results. For this, we have run several comparison tests with the shipment scenario defined in the SWS Challenge, a widely recognized use case for demonstration and comparison of Web service discovery techniques (`www.sws-challenge.org/`). Section 3 presents this evaluation in detail. Finally, Section 4 concludes the document.

2 Prototype

This section presents the prototype of the SDC technique. We here focus on the architecture and the central functionalities, referring to the SDC homepage for further technical details. The following first provides information on the availability of the prototype, then explains the architecture and design as well as the central algorithms of the SDC technique, and finally explains the technical realization of the prototype.

2.1 Availability

SDC Homepage: `http://members.deri.at/~michaels/software/sdc/`

Contact: Michael Stollberg, `http://members.deri.at/~michaels/`

Nature: Java application

Platform JDK 1.5.

Licensing LGPL (open source), copyright by DERI

Version 1.0, date: 26 March 2007.

Download `http://members.deri.at/~michaels/software/sdc/SDCstandalone.zip`

Source Control: CVS of the WSMX SourceForge project, web interface at
`http://wsmx.cvs.sourceforge.net/wsmx/components/discovery/src/main/org/deri/wsmx/discovery/caching/`

JavaDoc: `http://members.deri.at/~michaels/software/sdc/20070327/javadoc/`

Required Libraries: all included in the ”SDCstandalone.zip”archive

- WSMO4J – the WSMO API for Java
- Apache AXIS 2 – an open source engine for Web services
- Apache Log4J – an open source API for inserting logging statements into Java code.

2.2 Architecture and Design

The SDC technique is formally specified in a first-order logic setting [7] that – by purpose – is defined relatively independent of specific frameworks for Semantic Web services (e.g. OWL-S or WSMO). However, the discovery techniques are based on specific, sufficiently rich functional descriptions whose precise formal semantics allow to properly define the necessary matchmaking techniques. The following first explains the design considerations for the SDC prototype, and then presents the resulting technical architecture.

2.2.1 Design Considerations

Although the SDC technique is specified independent of a particular framework for Semantic Web services, it appears to be reasonable to adopt it to such a framework for realizing the prototype – in order to avoid the need of building the necessary infrastructure from scratch. The most appropriate choice for this is WSMO, as it complies with several design principles: in contrast to all other frameworks, WSMO defines goals

as a top level notion and therewith promotes the ontological differentiation of the requester and provider perspective [3]. Moreover, the exhaustive tooling infrastructure provided for WSMO can be used for the SDC prototype. In particular, the WSMO4J Java API for programmatically handling WSMO elements (see `wsmo4j.sourceforge.net/`) as well some of the base components of WSMX occur to be usable.

However, there are some differences between the WSMO specification and the one of SDC. At first, WSMO does not distinguish between goal templates and goal instances. Although this conception is implicitly made in several WSMO-based solutions and has been proposed as an extension [9], the distinction is not defined in WSMO and thus not supported in its tooling support. Secondly, the semantics of capabilities in WSMO is not precisely defined. However, this is a fundamental requirement for correctly defining formal matchmaking techniques for discovery. Thus, the SDC technique uses functional descriptions following the Abstract State Space model [4], a language independent model of Web services and the world they act in with precise formal semantics. As specified in [8] in detail, the functional descriptions for both goals and Web services used in SDC consist of:

(i) input variables IF that denote all required inputs
(ii) a precondition ϕ^{pre} that defines conditions on the start-state wherein IF occur as free variables
(iii) an effect ϕ^{eff} that defines conditions on the end-state wherein IF occur as free variables and the predicate out denotes the outputs.

The formal meaning is defined as so-called *implication semantics*: if, for a particular input binding in a specific start-state, ϕ^{pre} is satisfiable, then also ϕ^{eff} must be satisfiable in the end-state. In SDC, classical first-order logic [6] is used as the specification language for functional descriptions. This has been chosen because of its high expressiveness, and because it does not imply any constraints on the modeling of functional descriptions. However, this raises the third difference between WSMO and the SDC specification.

The WSMO framework comes along with an own specification language, the Web Service Modeling Language WSML [1]. This consists of of a conceptual part that allows to define WSMO elements (ontologies, goals, Web services, and mediators), and provides 5 variants of formal ontology specification languages.[1] Although not defined as an explicit variant, WSML FOL provides a first-order syntax with classical model-theoretic semantics.[2] This can be used as the specification language for the SDC prototype. However, it is not possible to use any other WSML variant because of the following reasons:

- *WSML Core* does not provide variables (needed for specifying functional descriptions)
- *WSML DL* is not expressive enough, as it does not support nominals (needed for defining input bindings for SDC functional descriptions)
- *WSML Flight* as well as *WSML Rule* have different formal semantics (minimal model semantics), and the respective reasoners do not provide the required subsumption reasoning.

[1] WSML variants (see specification [1]) :

1. *WSML Core* that relates to Description Logic Programs, the maximal intersection of Description Logics (DL) and Logic Programming (LP)
2. *WSML DL* as a Description Logic that is compatible with OWL-DL
3. *WSML Flight* as a restricted LP language
4. *WSML Rule* as fully qualified rule language
5. *WSML Full* that provides an umbrella of all variants as a first-order framework with auto-epistemic extensions.

[2] WSML FOL is defined as WSML Full without usage of the following symbols: ”naf”, ”!-”, ”ofType”, and without cardinality constraints [2].

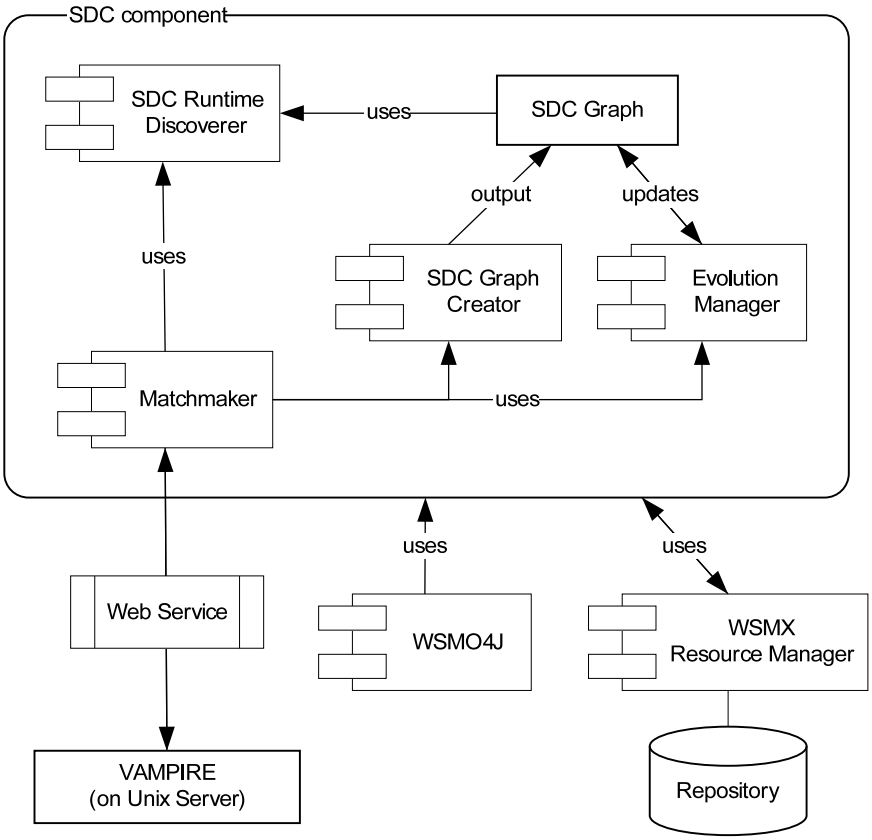


Figure 2: Architecture of SDC Prototype

2.2.2 Architecture of the SDC Prototype

The architecture for the SDC prototype results from the design considerations discussed above. It is implemented as a discovery component of the Web Service Execution Environment WSMX (see homepage: `www.wsmx.org`), the WSMO reference implementation. It uses WSML FOL as the specification language[3], and the following open source technologies: (1) the WSMO4J API for Java 5.0, (2) VAMPIRE 8.0, a first-order logic automated theorem prover for matchmaking [5], and (3) Apache AXIS 2 for invoking VAMPIRE via a Web Service (see `http://ws.apache.org/axis2/`).

Figure 2 provides an overview of the SDC prototype architecture. The overall SDC component consists of 3 main components: (1) the *SDC Graph Creator* performs the design time discovery and stores the result in the SDC Graph, the specific knowledge structure used by the SDC technique (see below in Section 2.2.3) . This is used by (2) the *SDC Runtime Discoverer* that performs the runtime Web service discovery, including the required functionalities for the goal formulation process; (3) the *Evolution Manager* maintains the SDC Graph in its changing environment, i.e. when a goal template or a Web service is added, removed, or modified. We shall explain and algorithms of these components below in Section 2.3.

[3]* The ontology, goal, and Web service descriptions for SDC are specified in WSML FOL, see footnote 2. The WSMO capability descriptions are translated to TPTP [10], a first-order logic syntax commonly used for automated theorem proving. The current translation is manual; an automation is under construction at the time of writing.

The necessary matchmaking techniques for the 3 main components are provided by the *Matchmaker*. This loads the relevant ontologies, goal, and Web service descriptions, associates these with respective TPTP representation, and dynamically adds the proof obligation for requested matchmaking. This proof obligation is checked with VAMPIRE; if it is provable, then the requested match is given. VAMPIRE is only available for Linux. In order to maintain the platform independence of the SDC prototype, it is invoked on a remote server via a Web service. We explain this in more detail below in section 2.4.

The other components used by the SDC prototype are *WSMO4J*, the Java API for WSMO (see above), and the *WSMX Resource Manager* that handles the storage and retrieval of WSMO elements in a repository (i.e. ontologies, goals, Web services, and mediators described in WSML).

2.2.3 SDC Graph Ontology

The heart of the SDC technique is the SDC Graph. This is the knowledge structure for caching design time discovery results that is used by all operations in the SDC technique: (1) as a search graph for finding the most appropriate goal template during the goal formulation process, (2) for optimization of the design time discovery, and – most importantly – (3) as the knowledge structure for enabling efficient runtime Web service discovery. The SDC graph is defined such that it provides the most appropriate knowledge structure for supporting all three usages. We briefly recall the definition, referring to [7] for the detailed formal specification and discussion.

The SDC graph consists of a *goal graph* that organizes goal templates in a subsumption hierarchy with respect to their semantic similarity, and a *discovery cache* that keeps minimal knowledge about the usability of the available Web services for each goal template. Two goal templates $\mathcal{G}_1, \mathcal{G}_2$ are considered to be *similar* if they have at least one common solution. This is expressed in terms of matching degrees between their formal functional descriptions (*exact*, *plugin*, *subsume*, *intersect*, *disjoint*); the same degrees are used to denote the functional usability of a Web service W for a goal template $\mathcal{G}$. Arcs in the SDC graph are directed connections of the form $d(source, target)$: the source is always a goal template; arcs with a goal template as the target constitute the goal graph, and those with a Web service as the target define the discovery cache; d describes the matching degree between the source and the target. Formally, the SDC graph is set of *directed acyclic graphs* (DAG). In each connected sub-graph, the inner nodes are similar goal templates and the leaf nodes are the usable Web services.

In the SDC prototype, the SDC Graph is defined as an ontology. The conceptual model is represented in the ontology schema, and an SDC graph for a specific application is handled as a knowledge base (i.e. and ontology instance store). This representation has been chosen in order to allow (1) re-use within other systems: the SDC Graph knowledge base can be used for interchanging knowledge about Web service discovery results, and (2) the integration with other Web service discovery techniques: e.g. ranking and quality-of-service discovery techniques can be performed on the basis of the SDC graph.

Listing 1 below shows the ontology schema of the SDC Graph in WSML, which is used with the SDC prototype. It is specified in *WSML Rule*, the most expressive WSML variant for which reasoning support exists at the time of writing. The concept **goal template** refers to the WSML goal description, and carries information of the *position* of a goal template in the SDC graph. The position can be *root*, i.e. the goal template that does not have any parents in the SDC graph, *child* when it has a parent, or *intersectionGT* that denotes the result of resolving an intersection similarity degree between two goal templates (see below, Section 2.3). The position attribute helps to deal with an SDC graph knowledge base in the algorithms. Following the definition from [8], the concept **goal instance** defines the corresponding goal template and the input binding β, which is represented as a WSML datatype with possibly multiple values. The matchmaking

degrees – used to denote the similarity of goal templates as well as the usability of a Web service for a goal template – are pre-defined as instances of the concept **matchingDegree**. The arcs in an SCD Graph are represented by the concepts **goalGraphArc** and **discoveryCacheArc**. Instances of the former constitute the goal graph, instances of the letter the discovery cache in a SDC Graph knowledge base.

Form the perspective of conceptual modelling, it would be more appropriate to model the SDC Graph arcs as tenary relations of the form $arc(source, target, degree)$. However, such constructs can not be represented in OWL. Thus, with respect to reusability, we decided to model them in terms of concepts and instances. This can be expressed in any ontology language (even RDF, when omitting the cardinality constraints). Besides, the arcs in the SDC graph are conceptually very close related to WSMO mediators, especially to GG Mediators (for the goal graph arcs) and WG Mediators (for the discovery cache arcs). The current WSMO mediator specification does not allow to state the matching degree – which is the most relevant information within the SDC graph. With respect to this, we decided to model SDC graph arcs as concepts with instances in order to not conflict with other usages of WSMO mediators. However, it is easy to generate GG and WGM mediators out of SDC graph instances.

```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-rule"

namespace
{_" http :// members.deri.at/~michaels/ontologies/SDContology.wsml#",
    wsml _"http :// www.wsmo.org/wsml/wsml-syntax#"
    }

ontology
_" http :// members.deri.at/~michaels/ontologies/SDContology.wsml"

concept goalTemplate
    description  impliesType wsml#goal
    position  impliesType goalGraphPosition

concept goalGraphPosition

instance root memberOf goalGraphPosition
instance child  memberOf goalGraphPosition
instance intersectionGT memberOf goalGraphPosition

concept goalInstance
    correspondingGoalTemplate impliesType goalTemplate
    inputs  impliesType (0 n) wsml#datatype

concept matchingDegree
instance exact memberOf matchingDegree
instance plugin memberOf matchingDegree
instance subsume memberOf matchingDegree
instance intersect  memberOf matchingDegree
instance disjoint  memberOf matchingDegree

concept goalGraphArc
    sourceGT impliesType (1 1) goalTemplate
    targetGT impliesType (1 1) goalTemplate
     similarity   impliesType (1 1) matchingDegree

concept discoveryCacheArc
    sourceGT impliesType (1 1) goalTemplate
    targetWS impliesType (1 1) wsml#webService
     usability  impliesType (1 1) matchingDegree
```

Listing 1: WSML Ontology Schema for the SDC Graph

2.3 Main Algorithms

In order to provide a basic understanding of the SDC technique, the following briefly outlines the central algorithms. As shown in Figure 2, the central functionalities are the creation of the SDC graph (in the *SDC Graph Creator*), the maintenance of the SDC graph in an dynamically changing environment (in the *Evolution Manager*), and the runtime Web service discovery for goal instances (in the *SDC Runtime Discoverer*). The detailed specification of the algorithms is provided in [7].

2.3.1 SDC Graph Creator

This component creates the SDC Graph for some given goal templates and Web services. To ensure that the SDC graph exposes the desired formal properties, this is realized in an incremental manner. Essentially, the creation commences with one goal template and then subsequently adds the other ones. Figure 3 provides an overview of the algorithm as a flow chart.

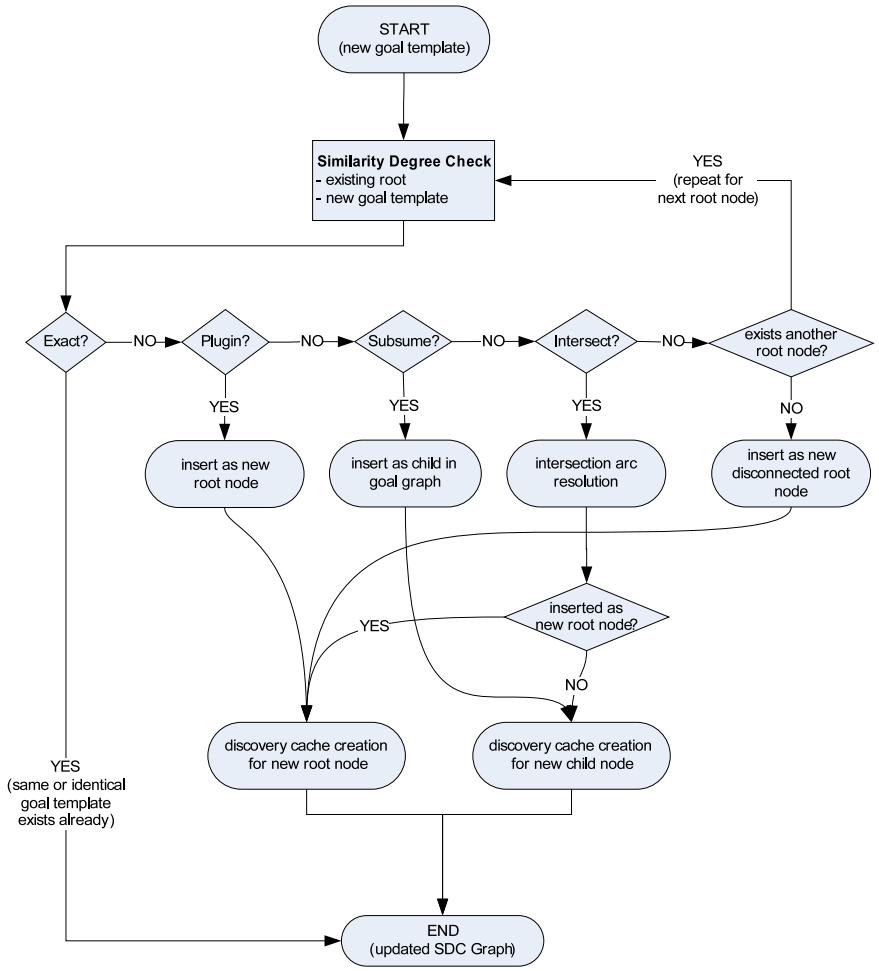


Figure 3: Overview of SDC Graph Creation Algorithm

The insertion of a new goal template into the SDC Graph commences with the creation of the goal graph. This means that the new goal template is inserted at the most appropriate position in the existing goal graph.

For this, at first the similarity degree between the new goal template and an existing root node is determined. If it is *exact*, then the new goal template is not added as an identical one exists already. If the degree *plugin*, then the new goal template is added as a new root node. If it is *subsume*, then the new goal template is inserted as a new child among the existing children of the inspected root goal template. For the case of an *intersect* degree, the so-called ”i-arc resolution”algorithm is invoked; we refer to [7] for the details of the insertion algorithms that are used under the *plugin*, *subsume*, and *intersect* similarity degrees. If none of the above cases is given, then the similarity degree check is repeated for the next root node. If no further root node exists, then the new goal template is inserted as a new, disconnected root node.

In the second step, the discovery cache is created – i.e. the relevant arcs between the newly inserted goal template and the existing Web services. Essentially, this performs the design time Web service discovery, i.e. the matchmaking of goal templates and Web services. This distinguishes two algorithms: if the position of the new goal template is *child*, then only those Web services that are usable for each of its parents can potentially be usable. We thus do not need to consider all available Web services, which enhances the efficiency of the discovery cache creation. If the position is *root*, we analogously can infer the usability degree for those Web services that are usable for each child node. However, also the other Web services need to be taken into account.

2.3.2 SDC Graph Evolution Manager

The purpose of the Evolution Manager is to maintain the formal properties of an existing SDC graph in the case that goal templates or Web services are added, removed, or modified. Figure 4 provides an overview of the algorithm as a flow chart; we refer to [7] for the detailed specification of the sub-procedures. While the changes on available Web services are invoked automatically via a notification mechanism by the Web service repository, changes on goal templates occur as manually performed maintenance operations (e.g. when outdated or false defined goal templates are removed by an administrator).

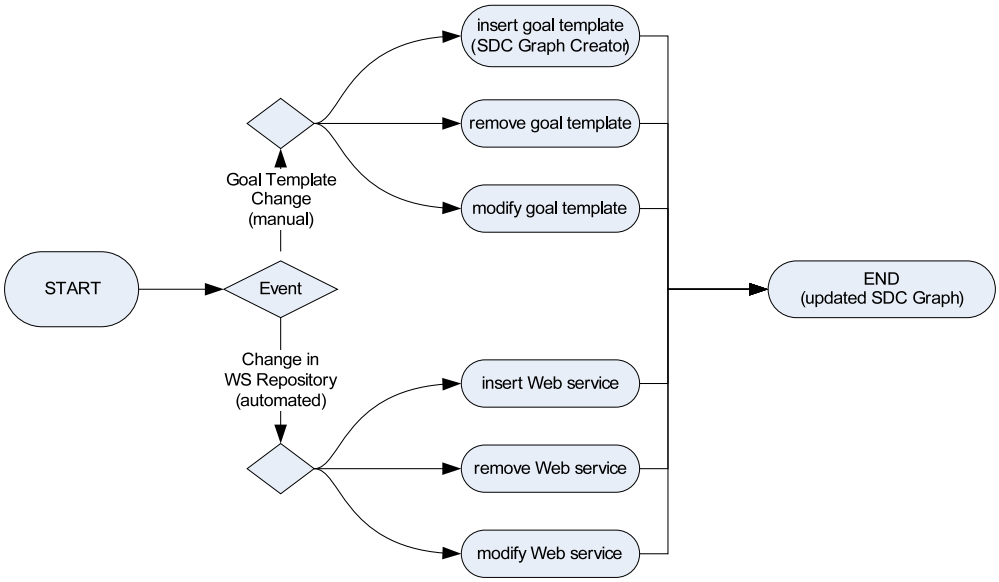


Figure 4: Overview of SDC Graph Evolution Management Algorithm

2.3.3 SDC Runtime Discoverer

The third central algorithm is the runtime Web service discovery. Its purpose is to find one usable Web service for a goal instance that has been created by a client. Although there might be several Web services that are usable under functional aspects, we only need to find and then invoke one for solving the goal instance. In order to minimize the computational costs, the SDC-enabled runtime discoverer makes extensive usage of the knowledge kept in the SDC graph.

Figure 5 provides an overview of the complete algorithm as a flow chart. At first, we must check whether the provided goal instance properly instantiates the specified corresponding goal template (conceptually, this belong to the goal formulation process). Then, we can try to find a usable Web service by *lookup*: if there is a Web service that is usable for the corresponding goal template under the *exact* or *plugin* degree, then it is also usable for the goal instance. Thus, we do not need to invoke the matchmaker, but instead answer the discovery task out of the SDC graph. If this is not successful, we then refine the goal instance by replacing the corresponding goal template with the most appropriate one. Then, we again can try to find a usable Web service via lookup (the probability for success is now higher). As the last option, those Web services are checked that are usable for the (refined) corresponding goal template under the *subsume* or *intersect* degree (the usability determination for the goal instance under these degrees requires matchmaking at runtime).

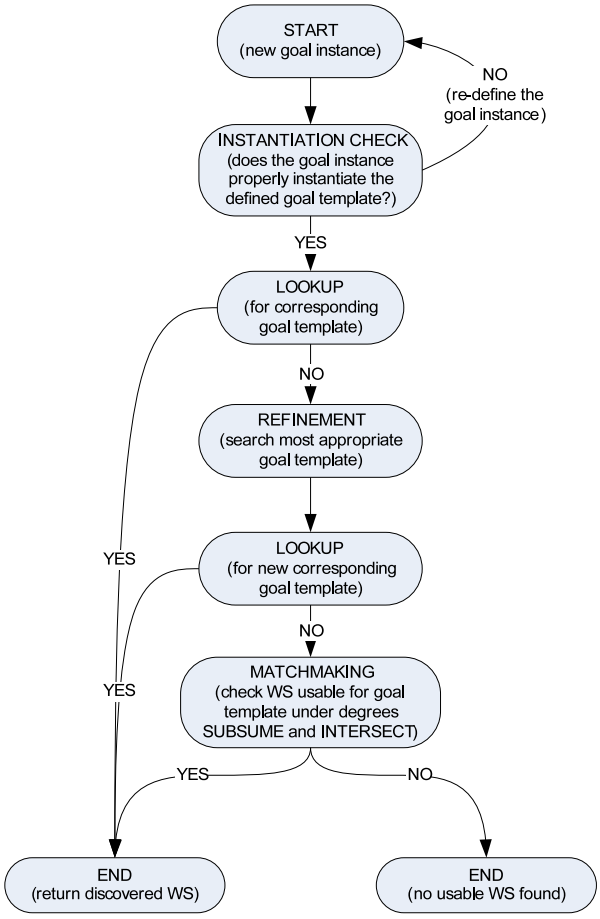


Figure 5: Overview of SDC Runtime Discovery Algorithm

2.4 Implementation Specification

We complete the presentation of the SDC prototype with the implementation specification. The following provides an overview of the technical realization. For further details, we refer to the source code available from the SDC homepage and the JavaDoc documentation (for links see Section 2.1).

As outlined above, the SDC prototype is implemented as a discovery component for WSMX. It uses the WSMO4J API for handling WSMO objects (goal and Web service descriptions, and in particular the SDC Graph as a WSML ontology knowledge base), and the FOL automated theorem prover VAMPIRE for matchmaking that is invoked via a Web service. The source classes are organized in two Java packages. The following explains their content, and Figures 6 and 7 show their structure as UML class diagrams.

Package: `org.deri.wsmx.discovery.caching`
This provides the main classes of the SDC Prototype, in particular the implementations of the three main algorithms outlined above. Its consists of the following classes (see Figure 6):

SDCResourceManager loads & stores WSML files on local machine

SDCGraphManager basic SDC Graph management as a WSML Rule Ontology, incl. the creation of Goal Templates and SDC Graph Arcs cache), and routines for handling the SDC Graph knowledge base

SDCGraphCreator implements the SDC Graph creation algorithm (see 2.3.1)

SDCGraphCreatorHelper provides helper methods for the SDCGraphCreator

SDCGraphEvolutionManager implements the SDC Graph evolution management algorithm (see 2.3.2)

GoalInstanceManager creation, management, and validation of goal instances

GoalInstanceSDCDiscoverer implements the SDC-enabled runtime Web service discovery (see 2.3.3)

Package: `org.deri.wsmx.discovery.caching.matchmaking`
This provides all facilities related to the necessary within the SDC technique, in particular the Web service for invoking VAMPIRE. Its consists of the following classes (see Figure 7):

Matchmaker defines all types of matchmaking requests needed for the SDC technique

- uses the Web service *VampireInvoker* to perform the matchmaking (see below)
- uses the *POGenerator* for generating the proof obligations (see below)
- the interface *Matchmaker* defines the method skeletons for all matchmaking requests needed for the SDC technique; other implementations of this interface may use different matchmakers

POGenerator generates the TPTP proof obligations on the client side

- a proof obligation is a logical statement [10] that represents a particular matchmaking request; this is to be proved by VAMPIRE
- the interface *POGenerator* defines the method skeletons for all types of proof obligations needed for the SDC technique

VampireInvoker Web service implementation class for invoking VAMPIRE on a remote server

- intermediately stores the TPTP proof obligation, invokes VAMPIRE for proving it, and returns the result (as a boolean)
- the Web service is publicly available at `http://138.232.65.138:8080/axis2/services/VampireInvoker?wsdl`; Appendix A provides the WSDL description

VampireInvokerStub client stub for the Web service *VampireInvoker* generated by AXIS 2

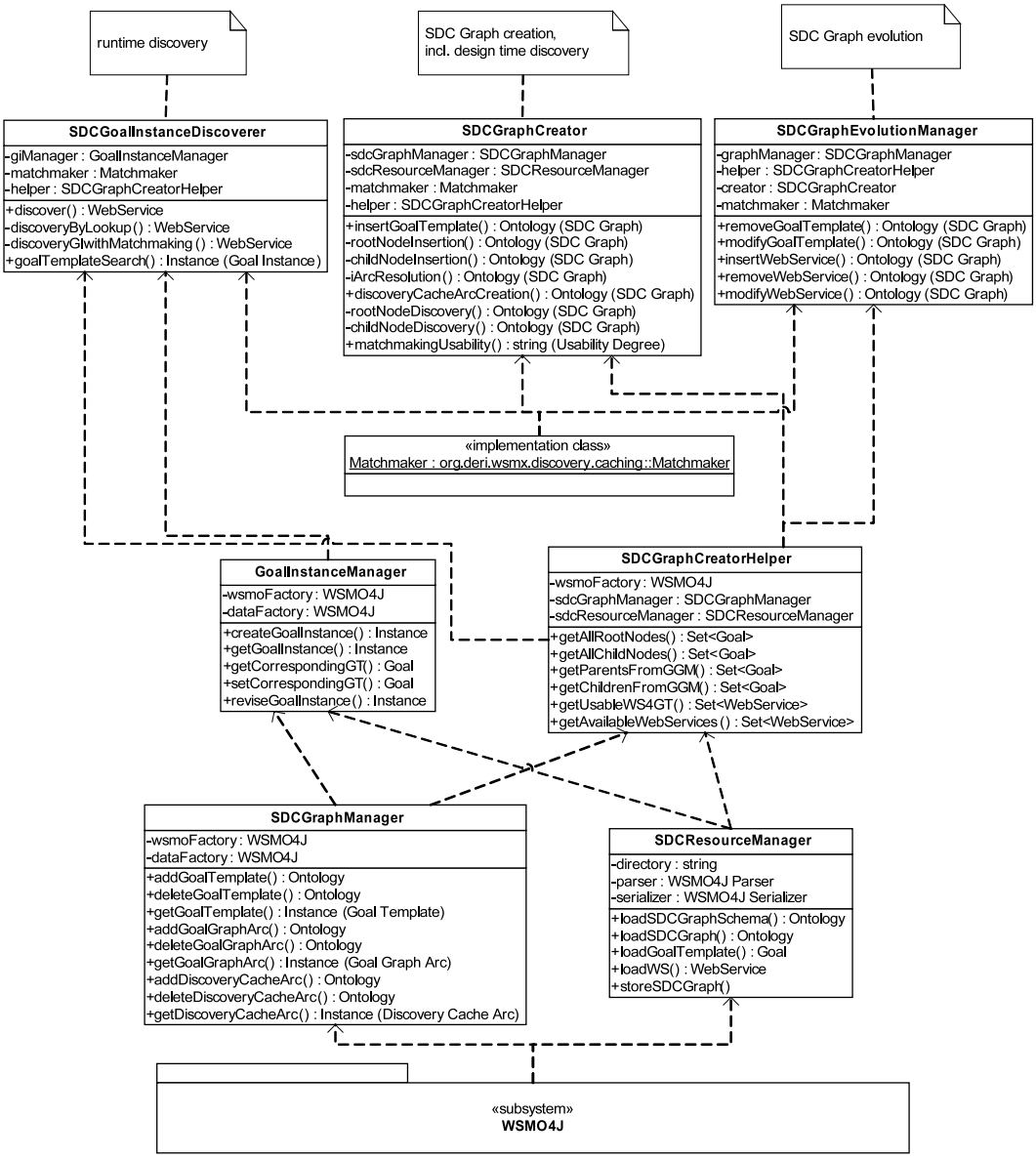


Figure 6: UML Class Diagram of SDC Prototype

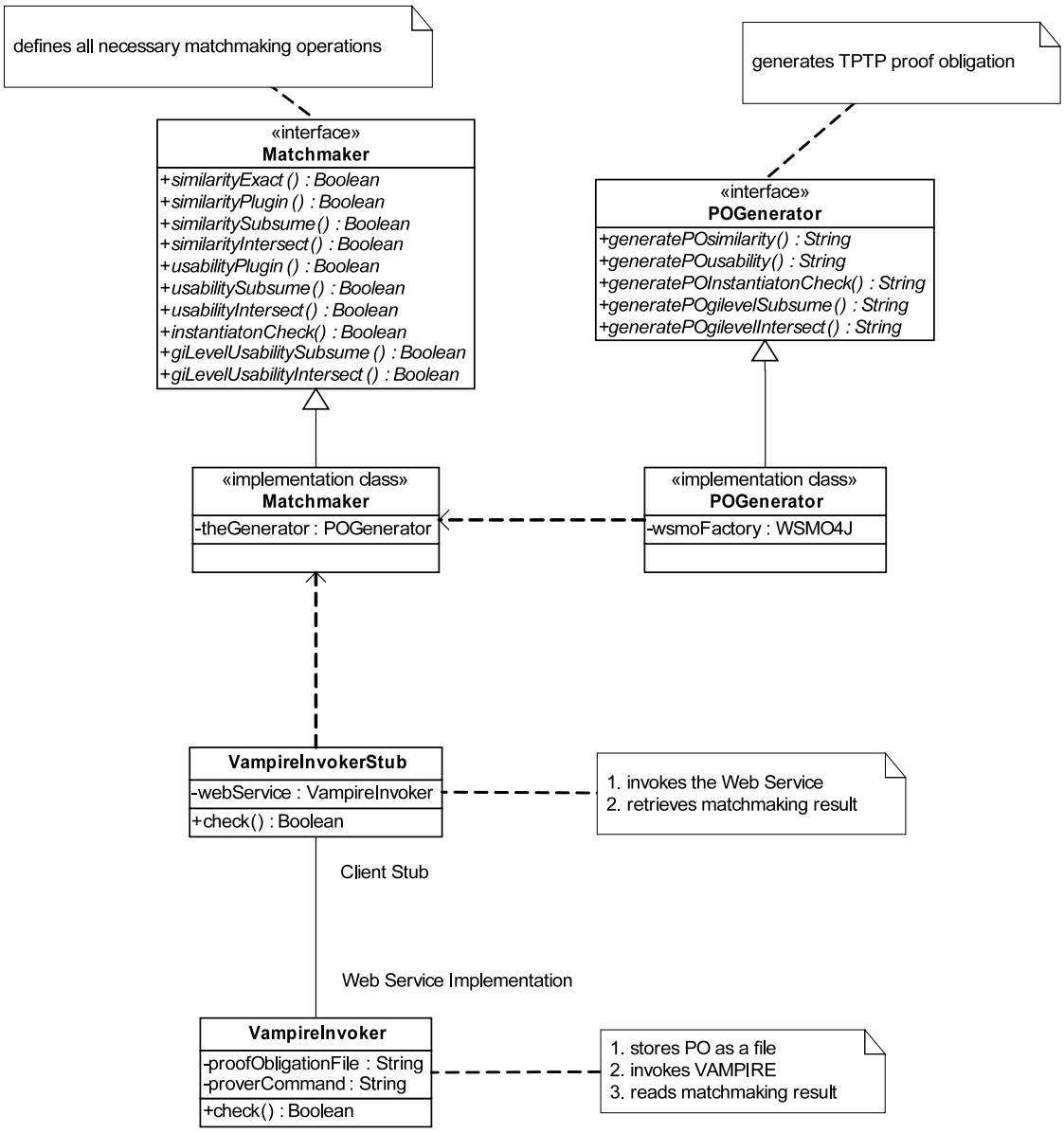


Figure 7: UML Class Diagram of Matchmaker used in SDC Prototype

3 Use Case and Evaluation

The main aim of the SDC technique is to enable efficient, scalable, and stable Web service discovery, therewith providing a sophisticated component for one of the central reasoning tasks for Semantic Web services. To evaluate the achievable efficiency increase, we perform and examine a comparison between the SDC-enabled runtime Web service discovery and a runtime discovery engine that applies the same matchmaking techniques but does not make use of the knowledge kept in the SDC Graph. For this comparison, we use the shipment scenario that has been defined in the Semantic Web Services Challenge – a widely accepted initiative for demonstration and comparison of semantically enabled discovery techniques.

This section explains the evaluation setting, the methodology, and discuss the results. We first explain the use case setting, and how this is mapped to the conceptual discovery framework of SDC. Then, we explain the evaluation methodology, present the results of the comparison test along with adequate statistical techniques, and finally discuss these with respect to the relevance and scientific contributions of this research.

3.1 Use Case

For demonstration and evaluation we apply the shipment scenario that is defined within the Semantic Web Services Challenge (short: SWSC), (`www.sws-challenge.org/`). This challenge provides different scenario settings that are designed to present and compare different SWS technologies, among them discovery, composition, and mediation. Each scenario describes a problem setting on the basis of real-world settings (e.g. really existing (Web) services, and). The challenge is to properly map the scenario description to the conceptual framework that underlies a specific technical solution, and then to demonstrate how the developed techniques handle the problem.

3.1.1 SWS Challenge Shipment Scenario

The SWSC shipment scenario is the one that is particularly dedicated to Web service discovery, with a primary focus on functional aspects. Thus, it is the most appropriate SWSC use case scenario. Besides using a widely accepted use case for demonstration and evaluation, this choice also allows to compare the SDC technique and its underlying conceptual framework with other solutions that have been presented for this scenario (see `http://sws-challenge.org/wiki/index.php/Scenarios`).

The SWSC shipment scenario is concerned with shipping packages from a Sender to a Receiver. There are five Web services defined (with WSDL descriptions); each of these offer the shipment of packages from the USA to a specific list of countries in the world. Moreover, each Web service has specific conditions regarding the price of the shipment service (not the usage of a Web service, but the price for using the real-world service that is accessible via a Web service) and further conditions (e.g. maximum weight, delivery time, etc.). Then, several potential client requests along with the expected discovery result are defined. These requests are grouped with respect to different aspects that should be taken into consideration for Web service discovery, e.g.: (A) destination only, (B) destination and weight, (C) destination, weight and price. A discoverer is expected to provide a technical solution that complies with the expected discovery results.

3.1.2 Mapping to SDC

The first step for demonstrating and evaluating the SDC technique within the SWSC shipment example is to map the scenario description to the conceptual framework that underlies the SDC approach. This means to model the goals and Web services, and the necessary domain ontologies. The following explains this. Due to

space limitations, we restrict this explanation to the central conceptual aspects. The complete descriptions of all relevant resources are available in the archive *resourcesSWSC.zip*, available for download from the SDC homepage (see `http://members.deri.at/~michaels/software/sdc/`).

Scope and Elements

The purpose of this use case demonstration is to examine the increase in the computational efficiency of the Web service discovery process that is achievable with the SDC technique. For this, the behavior of different discovery engines for large numbers of Web services appears to be more relevant than the complexity of the formally described requested and provided functionalities. Nevertheless, we of course need to have functional descriptions of goals and Web services that properly reflect the SWSC shipment scenario.

With respect to this, we define the scope of functional descriptions for goals and Web services for the demonstration to contain three aspects: the location of the sender, the location of the receiver, and the weight of the package that is to be shipped. This covers requests of the types (A) and (B) as defined in the SWSC shipment scenario (see above). We do not consider the shipping price, because this would significantly increase the reasoning complexity (especially considering the fact that TPTP does not support numeric arithmetics). Moreover, in this scenario the shipping price occurs to be an aspect for selection or ranking mechanisms (e.g. for finding the cheapest shipment service for a particular request). The functional descriptions for the chosen scope have a relatively low logical complexity, which results in relatively fast processing times for individual matchmaking operations.

Table 1 shows the 5 Web services of the SWSC challenge with their provided functionality under the chosen scope. For example, *Muller* ships packages of a maximal weight of 50 lbs from the USA to countries located in Africa, Europe, North America, and Asia. The package weight is modeled in terms of weight classes that range from ”light“ to ”heavy“ in gaits of 10 lbs. There is another Web service that – by purpose – is not usable for solving any request from the scenario. In the SDC conceptual model, a goal template describes a request for shipping a package on a generic level, e.g. from a country to another country. A goal instance then specifies the concrete sender and receiver address; we explain this below in more detail.

Table 1: Overview of Web Services used in Demonstration

Web Services	SenderLoc	ReceiverLoc	WeightClass
wsMuller	USA	AF,EU,NA,AS	w50lq
wsRacer	USA	AF,EU,NA,SA,AS,OC	w70lq
wsRunner	USA	EU,AS,SA,OC	heavy
wsWalker	USA	AF,EU,NA,SA,AS,OC	w50lq
wsWeasel	USA	USA	heavy
wsOther	also has 3 inputs, but is for renting a car		

Domain Ontologies

The basis for semantic descriptions of goals and Web services are domain ontologies that formally define the terminology and background knowledge. For this demonstration, we define two ontologies; the complete specifications in WSML are provided in Appendix B. We remark that these ontologies are defined for academic demonstration purpose of the SDC technique.

1. **Location Ontology:** defines continents, countries, and cities with respect to their geographic locality
2. **Shipment Ontology:** defines the relevant background knowledge for shipment; central concepts: shipment order, sender, receiver, package, price, weight classes

Goal and Web Service Descriptions

The functionalities requested by goals and provided by Web services are formally described; these serve as the basis for matchmaking. As outlined above, the SDC technology uses WSML as the specification language with a translation to TPTP, the FOL input syntax supported by VAMPIRE [10]. At the time of writing, the translation is performed manually; an automation of this is under construction.

The following illustrates this for the description of the Web Service *Muller* mentioned above. Listing 2 shows the functional description as a WSML capability. The specified WSML variant is WSML Full; however, it actually is WSML FOL (see footnote 2). Essentially, this description states that if a shipment request is provided with a sender address in the USA and a receiver address in a country that is located in one of the supported continents (this is the *precondition*), then an order for the shipment of a package with the provided information will be obtained (*postcondition*).

```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-full"
webService _"http ://.../ swsc-shipment/webservices/wsMuller.wsml"
 importsOntology {
   _" http  ://.../  swsc-shipment/ontologies/shipment.wsml#",
   _" http  ://.../  swsc-shipment/ontologies/location.wsml#"}
 capability wsMullerCapability
  sharedVariables {?SendLoc,?RecLoc,?Package,?Weight,?O}
  precondition definedBy
    forAll  (?Sender,?Receiver).
     ?Sender[address hasValue ?SendLoc] memberOf sho#Sender and
          ?SendLoc[locatedIn hasValue loc#US] memberOf loc#Location and
     ?Receiver[address hasValue ?RecLoc] memberOf sho#Receiver and
          (?RecLoc[locatedIn hasValue loc#Africa] memberOf loc#Location or
           ?RecLoc[locatedIn hasValue loc#Europe] memberOf loc#Location or
           ?RecLoc[locatedIn hasValue loc#NorthAmerica] memberOf loc#Location or
           ?RecLoc[locatedIn hasValue loc#Asia] memberOf loc#Location) and
     ?Package[weight hasValue ?Weight] memberOf sho#Package and
     ?Weight[includedIn hasValue sho#w50lq].
  postcondition definedBy
   ?O[
     from hasValue ?SendLoc,
     to  hasValue ?RecLoc,
     item hasValue ?Package ,
     price  hasValue thePrice] memberOf sho#shipmentOrder.
```

Listing 2: Functional Description of Web service Muller in WSML FOL

Listing 3 below shows the corresponding functional description in TPTP. It specifies the same conditions, and, in addition, follows the structure of functional description as defined in [8]; here, we use the conjunctive semantics. The 4 input variables as well as the output variable ”O“ are universally quantified; the predicate `ws(...)` allows to access the description within a proof obligation. Although there is no formally defined semantics for WSML capabilities, the WSML representation can be used with the same formal meaning. For this, the WSML elements need to be translated into the TPTP structure: the shared variables become the input variables, and the relationship between the precondition and postcondition needs to be explicated. This is considered to be future work for the implementation of the SDC matchmaker.

```
input_formula(wsMuller,axiom,(
  ! [SendLoc,RecLoc,Package,Weight,O] : (
  ws(SendLoc,RecLoc,Package,Weight,O) <=> (
  % PRECONDITION
     (
       locatedIn(SendLoc, usa) &
       (
```

```
        locatedIn(RecLoc, africa)  |
        locatedIn(RecLoc, europe) |
        locatedIn(RecLoc, northAmerica) |
        locatedIn(RecLoc, asia)
      )
      & package(Package) & weight(Package,Weight)
      & includedIn(Weight,w50lq)
    )
  &
  % EFFECT
    (
      shipmentOrder(O) &
      from(SendLoc) &
      to(RecLoc) &
      item(O,Package) &
      price(O,price)
    )
  ) )
)).
```

Listing 3: Functional Description of Web service Muller in TPTP

Goal Templates and SDC Graph

The final aspect for mapping the SWSC shipment scenario to the SDC conceptual model is the SDC graph, i.e. the goal templates and the usability of the available Web services for these.

Figure 8 shows the goal graph for the SWSC shipment scenario, respectively the snapshot thereof that is actually used for the demonstration. The goal template *gtRoot* describes the objective of shipping a package of any weight from anywhere in the world to anywhere in the world; this is most general goal template, and hence is allocated as the root node of the SDC graph. The subsequent levels are constituted by the following structure: the level below the root restrict the sender and receiver locations to continents, the next level restricts these to countries, and the lowest level distinguishes the weight classes.

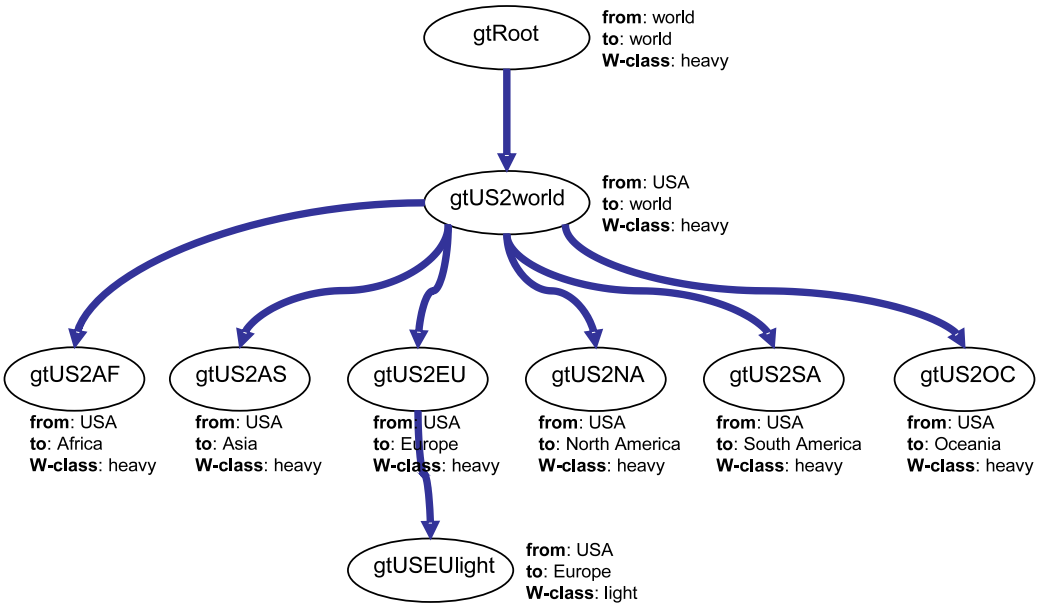


Figure 8: Overview of the Goal Graph for the SWSC Shipment Scenario

The goal graph shown above has only subsume arcs, and each child has exactly one parent. The SDC graph that will be dynamically created during the life time of the SWSC application will emerge towards this structure; in intermediate states also intersection goal templates can exist. Regarding the creation of goal templates: these can either be created manually, or they can be learned from concrete requests. Imagine a client specifies a goal instance for shipping a package of 1.5 lbs from San Francisco to Berlin. The formal description can be lifted to a more general level, e.g. to ”from USA to Germany with weight class light“. Subsequently, we can construct the root goal template by upwardly inspecting of the taxonomy of the used domain ontologies. We refer to [7] for a more detailed discussion on these issues.

The second part of the SDC graph is concerned with the usability of the Web services. Table 2 provides an overview of the usability degrees of the Web services for each of the goal templates defined above in Figure 8. This information is explicated in terms of discovery cache arcs in the SDC graph knowledge base. We observe that for the goal templates located at the upper levels of the goal graph, the most common usability degree is *subsume*. For those ones at the middle level we find all possible usability degrees, and for those at the lowest level the most common usability degree is *plugin*. It is to remark that the discovery cache arc between ”gtUS2EUlight”and ”wsRunner”is omitted in the SDC graph. The reason is that because wsRunner is usable under the plugin degree for a parent node in the goal graph (namely: ”gtUS2EU”), we can infer that its usability degree for ”gtUS2EUlight”is plugin. We do not store such redundant information in the SDC graph in order to keep it minimal. Besides, the ”wsOther”is not usable for any goal template; this Web service is used within the comparison test (see below).

Table 2: Overview of Usability Degrees of Web Services in SWSC Shipment Scenario

Usability	wsMuller	wsRacer	wsRunner	wsWalker	wsWeasel	wsOther
level 1						
gtRoot	subsume	subsume	subsume	subsume	subsume	X
level 2						
gtUS2world	subsume	subsume	subsume	subsume	subsume	X
level 3						
gtUS2AF	intersect	intersect	X	intersect	X	X
gtUS2AS	intersect	intersect	plugin	intersect	X	X
gtUS2EU	intersect	intersect	plugin	intersect	X	X
gtUS2NA	intersect	intersect	X	intersect	subsume	X
gtUS2OC	X	intersect	plugin	intersect	X	X
gtUS2SA	X	intersect	plugin	intersect	X	X
level 4						
gtUS2EUlight	plugin	plugin	plugin (omitted)	plugin	X	X

3.2 Evaluation

After explaining the SWSC shipment scenario and its mapping into the SDC model, we now turn towards the evaluation of the SDC technique. The main aspect of interest is the runtime discovery, i.e. to find a usable Web service for a concrete client request that is formulated in terms of a goal instance. A good discovery engine should perform this operation **efficiently** (within a minimal time), and it should be **scalable** (stay operational for large numbers of available Web services). This is what is relevant in real-world scenarios.

In order to evaluate the improvements achievable with SDC in this respect, we define 10 goal instances for the SWSC scenario and compare the time for successful runtime discovery between the SDC-enabled runtime Web service discovery and a runtime discovery engine that applies the same matchmaking techniques but does not make use of the knowledge kept in the SDC Graph. The following explains the methodology for evaluating this by the means of a performance comparison, presents the results, and upon these discusses the relevance and contributions of this research.

3.2.1 Methodology

Table 3 provides an overview of the 10 goal instances that are used for the comparison. Following the structure of the client requests defined in the SWSC scenario description, these are defined such that different

Table 3: Overview of the 10 Goal Instances used in Performance Comparison Test

NO / ID	GT	from	to	weight	usable WS
1	gtUS2AF	california	tunis	1	wsMuller wsRacer wsWalker
2	null	california	luxembourgCity	1.5	wsMuller wsRacer wsRunner wsWalker
3	gtUS2AF	california	tunis	50.5	wsRacer
4	gtUS2EU	california	bristol	40	wsRacer wsRunner
5	gtUS2NA	california	newYorkCity	5.5	wsMuller wsRacer wsWalker wsWeasel
6	gtUS2world	california	berlin	60	wsRunner
7	gtUS2world	california	sydney	17.3	wsRacer wsRunner wsWalker
8	gtUS2SA	california	quito	27.58	wsRacer wsRunner wsWalker
9	gtUS2AS	california	beijing	57.8	wsRacer wsRunner
10	gtUS2EUlight	california	amsterdam	9.99	wsMuller wsRacer wsRunner wsWalker

parts of the SDC runtime discoverer need to be applied, e.g. the refinement of the corresponding goal template (no. 2,6,7), discovery by lookup (no. 2,6,7,8,9,10), as well as additional matchmaking (1,3,5).

As the comparison discovery engine, we implemented a second runtime discoverer that uses the same matchmaking facilities as the SDC-enabled runtime discovery but does not make use of the knowledge kept in the SDC graph. This engine performs two operations in order to find a usable Web service for a given goal instance: (1) the refinement of the corresponding goal template, and (2) checking the basic goal instance level matching requirement as defined in [8]. The former operation is needed in order to ensure that the client objective is properly described: if the a priori specified corresponding goal template is not the most appropriate one, then it must be refined (namely for goal instances 2, 6, and 7). For the second operation, the goal instance level matching is defined to be given if the formula $\Omega_{\mathcal{A}} \cup \{[\phi^{\mathcal{D}_{\mathcal{G}}}]_{\beta}, [\phi^{\mathcal{D}_W}]_{\beta}\}$ must be satisfiable. In plain words: a match is given if the functional descriptions of both the corresponding goal template and Web service are satisfiable under the input binding defined in the goal instance, with respect to the domain ontology. We shall refer to this runtime discovery engine as the *non-SDC discoverer* in the following.

The comparison consists of tests runs between the SDC-discoverer and the non-SDC discoverer for all 10 goal templates, each for the following number of available Web services: 10, 20, 50, 100, 200, 500, 1000, 1500 and 2000. These numbers have been chosen with respect to the expectable size of real-world SOA applications. Although there might be more than 2000 available Web services, we shall see that this is sufficient to obtain valuable comparison data. In order to ensure the statistical significance of the data, we repeat each test 50 times. The set of available Web services always contains the 5 Web services from the SWSC scenario, so that both discoverers always will terminate with a positive discovery result. The non-SDC discoverer subsequently checks the available Web services in a randomized order; it stops as soon as one usable Web service has been found. It therewith provides a runtime discovery engine that does not use any performance optimization, which simulates most of the existing engines for semantically enabled Web service discovery.[4]

3.2.2 Results

The following presents the results of the comparison test as described above. We here present the overall comparison results and discuss their meaning. A more detailed overview of the test data is provided in Appendix C.

In order to properly evaluate and discuss the obtained comparison data, we apply standard statistical notions (in particular: the median and the standard deviation; see Appendix C for the definitions). The most important result is the performance of the SDC and the non-SDC runtime discoverer, i.e. the time that is needed to find a usable Web service for a given goal instance. Figure 9 shows the performance comparison charts, presenting the average of all test runs for all 10 goal instances. From this, we can observe the following qualities:

1. computational efficiency: *The SDC discoverer is, in average, faster than the non-SDC discoverer.* The reason is that in certain cases the SDC discoverer can find a usable Web service without needed of a matchmaker. In the SWSC scenario, this is given for the goal instances 4, 8, 9, and 10. When examining the comparison results, we observe that only for one goal instance (no. 7) the non-SDC discoverer is faster for the test cases of 10 and 20 available Web services.

[4]The Java resources for the comparison test are included in the download version of the SDC prototype (see details in Section 2.1): the class *SWSCComparison* provides the implementation of the non-SDC discoverer and an invocation method for the SDC discoverer that measures the time, and the JUnit test *SWCComparisonTester* provides a skeleton for automated test runs that writes the test data into CSV files along with a log.

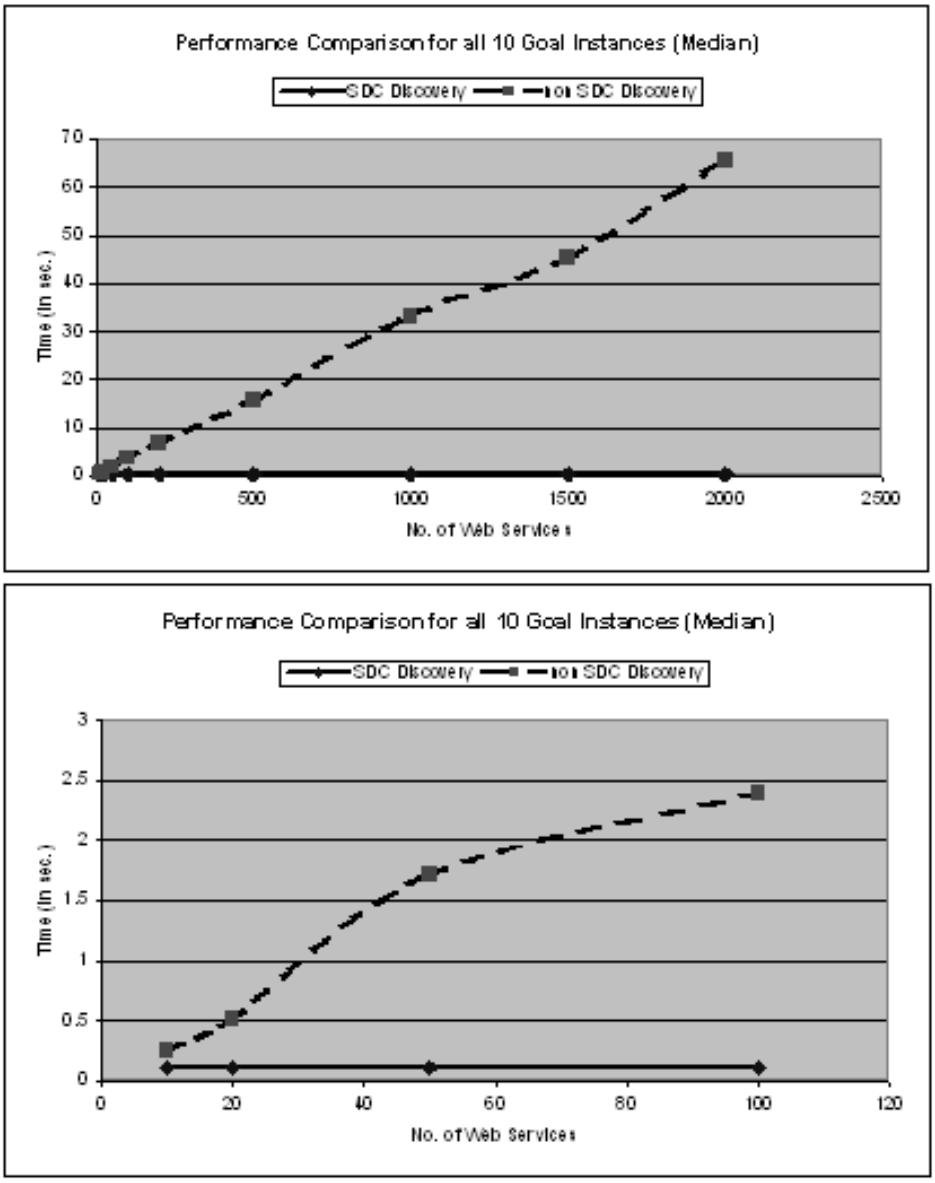


Figure 9: Performance Comparison Charts

2. scalability: *For an increasing number of Web services, the time required by the SDC discovery stays the same while for the non-SDC discoverer it grows proportionally.*
The reason is the pre-filtering of the SDC discovery: only those Web services that are usable for the corresponding goal template need to be considered at runtime. Moreover, we see that the average time required by the SDC discoverer is constantly under 1 second, independent of the number of available Web services. In contrast, the non SDC discovery requires, in average for the SWSC use case, more than 1 minute for 2000 available Web services; this occurs to be a not acceptable performance in real world scenarios.

Apart from the increases in computational efficiency and scalability, the evaluation data expose another quality that occurs to be important for reliable Web service discovery engines: **stability**. This is concerned with behavior of the Web service discovery engine under several calls, which becomes important when the discoverer is used as a component within more complex Web service execution environments. Figure 10 shows the variance of the 50 individual test runs for goal instance 1 for 10 available Web services. We observe that the SDC discoverer varies between 93 and 188 milliseconds, while the non-SDC discoverer ranges from 78 to 797 milliseconds. This difference gets bigger with an increasing number of available Web services (e.g. for 2000 Web services, the non-SDC discoverer varies between 1.4 and 175.5 seconds). The smaller the variation, the better the behavior of a discovery component can be estimated within an overall SWS architecture.

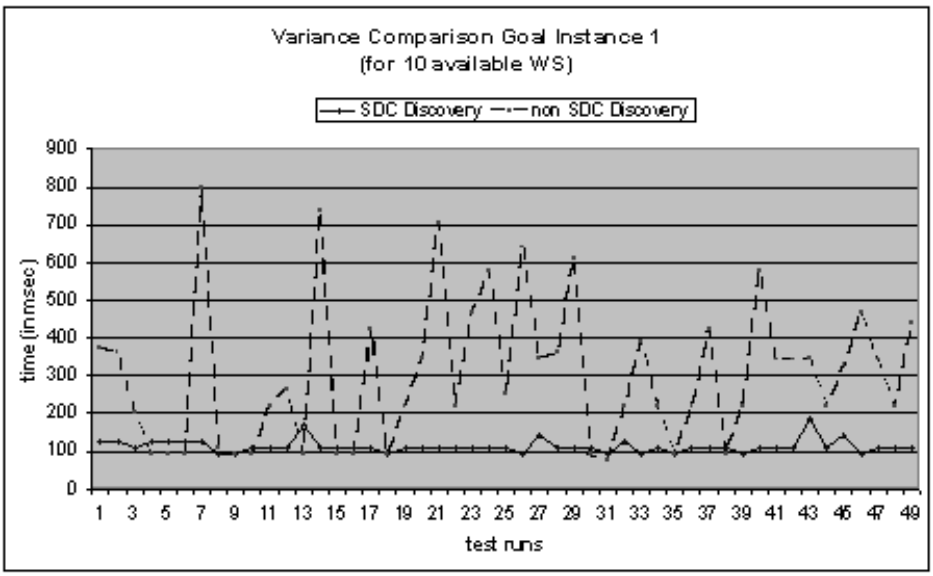


Figure 10: Performance Comparison Charts

3.2.3 Discussion of Impact

We complete the evaluation with an outline on the relevance and expectable impact for the SDC technique. We depict three aspects here that will be further elaborated in the future.

1. The need for efficient, scalable, and stable Web Service discovery engines
Already existing SOA systems contain thousands of Web services – e.g. the one from VERIZON has nearly 4000. It is commonly expected that the number of Web services in real world applications will become significantly higher. The basic requirement for a automated discoverer component is that it finds the right Web services (correctness of the discovery result), and that it is capable of dealing with the number of available Web services (i.e. scalability). One may argue that there is an improvement if a discovery engine does this – even if it can take 5 minutes.

However, in accordance to the common vision among the SWS research community, we understand the aim of SWS technologies to automate or at least mechanize the Web service usage process to a high extent. Discovery is one of the central reasoning tasks for overall SWS system. It is not only for detecting directly

usable Web services, but provides a basic component for more complex technologies like Web service composition (for finding composition candidates out of the available Web services), or for dynamic business process management systems that need to discover and combine several hundreds of Web services. In such systems, the Web service discoverer becomes one component among others (although it will be a central one). In order to allow the proper usage of a Web service discovery engine as a software component, its properties on efficiency, scalability, and stability become essential quality criteria. In the evaluation we have shown that the SDC technique exposes these qualities in a significantly better manner than the non-SDC discoverer that simulates existing Web service discovery engines.

2. Integration of other Web Service discovery techniques
Within this research, we have only considered functional aspects of Web services for determining their usability for solving a goal. However, there are of course other relevant aspects for discovery, in particular selection (that is to filter the list of functionally usable Web services) and ranking techniques (to determine a priority list for discovered Web services).

Naturally, the SDC-enabled discovery can be integrated with such techniques. We thereby keep the primary focus on functional aspects: if a Web service does not provide the functionality that is capable of solving a request, then it does not matter whether it is better then some other Web service. However, the interesting aspect is the behavior of a discovery engine that, apart from the functional aspects, allow to perform selection and ranking.

Before a selection or a ranking technique can be applied, the functionally usable Web services out of the available ones must be determined. If we want to do this for a concrete client request (i.e. a goal instance), then the functional discovery needs to return all usable Web services – not just one as we have done in the comparison. For this, an SDC-enabled functional discoverer only needs to check those Web services that are usable for the corresponding goal template, and it may can do this by lookup (i.e. without matchmaking). The non-SDC discoverer would necessarily have to inspect *all* available Web services – to ensure that the complete set of usable ones will be found. Thus, the number of matchmaking operations will be the same as the number of available Web services. Taking the computational costs for matchmaking from the evaluation in the SWSC scenario above, this would be about 115 ms * 2000 = 230 seconds for 2000 Web services. In consequence, it can be expected that the differences between the SDC-enabled Web service discovery and the comparison engine becomes even more significant.

3. Adaptability of the SDC technique
The SDC technique is specified independently of particular frameworks for Semantic Web Services. Thus, it can in general be adopted to any SOA system – with a few modifications on the concepts and definitions.

One may consider the usage of FOL as the specification language as a weakness, in particular with respect to the problem of unsatisfiability. This choice has been made because it allows to exactly model and handle functional descriptions of goals and Web services as they are specified in the underlying conceptual model – which, to our understanding, most adequately represents the real world and formalizes the intuitive understanding of the relevant concepts (goals, Web services, instantiations, the meaning of a match, and the matchmaking degrees).

The adoption of this model to decidable subsets of FOL (and, in consequence, to existing ontology languages) is not possible without the loss of precision. Nevertheless, a partial mapping of the SDC model to DL- or LP-based specification languages is of course possible, and therewith some of the benefits of the SDC technique can become applicable. Moreover, one can imagine that the concept of caching Web service usages for client requests can also be applied in SOA systems that do not (yet) use any semantic techniques.

4 Conclusions and Future Work

The SDC technique presents a novel approach for increasing the efficiency and scalability of Web service discovery by adopting the concept of caching – a well established means for performance engineering. Its theoretic foundations define the knowledge structure and algorithms for caching of Web service discovery. Although specified in a first-order logic setting and the prototype is implemented within the WSMO framework, the principle can be adopted to any framework for semantically describing Web services. The evaluation shows that a significant increase in the efficiency and scalability of Web service discovery can be achieved. Therewith, the SDC technique provides a contribution to the realization of semantically enabled SOA technology with respect to the requirements arising in real world scenarios.

The current SDC prototype is operational. The future developments will be concerned with its integration into semantically enabled SOA systems, in particular the WSMX reference implementation. For this, the following extensions are planned:

- visualization and goal instance creation support in WSMT, the graphical user interface for WSMX
- automation of the translation from WSML to TPTP
- adaptation of the SDC technique to other WSML variants, in particular the DL and LP branches
- integration into the WSMX executions semantics, incl.
  - distinction of goal templates and goal instances
  - design time and runtime discovery
  - integration of other discovery aspects (e.g. non-functional and quality-of-service aspects)

References

[1] J. de Bruijn, H. Lausen, R. Krummenacher, A. Polleres, L. Predoiu, M. Kifer, and D. Fensel. The Web Service Modeling Language WSML. Deliverable D16.1 final draft 05 Oct 2005, WSML Working Group, 2005. Available at: http://www.wsmo.org/TR/d16/d16.1/v0.2/.

[2] J. de Bruijn and S. Heymans. WSML Ontology Semantics. WSML Deliverable D28.3 Final Draft, 2006. available at: http://www.wsmo.org/TR/d28/d28.3/.

[3] D. Fensel, H. Lausen, A. Polleres, J. de Bruijn, M. Stollberg, D. Roman, and J. Domigue. *Enabling Semantic Web Services. The Web Service Modeling Ontology*. Springer, 2006.

[4] U. Keller, H. Lausen, and M. Stollberg. On the Semantics of Funtional Descriptions of Web Services. In *Proceedings of the 3rd European Semantic Web Conference (ESWC 2006), Montenegro*, 2006.

[5] A. Riazanov and A. Voronkov. The Design and Implementation of VAMPIRE. *AI Communications*, 15(2):91–110, 2002. Special Issue on CASC.

[6] R. M. Smullyan. *First Order Logic*. Springer, 1968.

[7] M. Stollberg. Semantic Discovery Caching: Specification. Technical Report DERI-2007-02-03, DERI, 2007.

[8] M. Stollberg, U. Keller, H. Lausen, and S. Heymans. Two-phase Web Service Discovery based on Rich Functional Descriptions. In *Proc. 4th European Semantic Web Conference (ESWC 2007)*, 2007.

[9] M. Stollberg and B. Norton. A Refined Goal Model for Semantic Web Services. In *Proc. of the 2nd International Conference on Internet and Web Applications and Services (ICIW 2007), Mauritius*, 2007.

[10] G. Sutcliffe and C.B. Suttner. The TPTP Problem Library: CNF Release v1.2.1. *Journal of Automated Reasoning*, 21(2):177–203, 1998.

[11] W. Voss. *Taschenbuch der Statistik*. Hanser Fachbuchverlag, 2. edition, 2003.

APPENDIX

A VampireInvoker Web Service – WSDL

```
<wsdl:definitions xmlns:axis2="http://matchmaking.caching.discovery.wsmx.deri.org"
  xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
  xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
  xmlns:ns0="http://matchmaking.caching.discovery.wsmx.deri.org/xsd"
  xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
  xmlns:ns1="http://org.apache.axis2/xsd"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  targetNamespace="http://matchmaking.caching.discovery.wsmx.deri.org">
<wsdl:documentation>
runs a proof obligation that is provided as input
</wsdl:documentation>
<wsdl:types>
 <xs:schema xmlns:ns="http://matchmaking.caching.discovery.wsmx.deri.org/xsd"
    attributeFormDefault="qualified" elementFormDefault="qualified"
    targetNamespace="http://matchmaking.caching.discovery.wsmx.deri.org/xsd">
 <xs:element name="check"> <xs:complexType> <xs:sequence>
  <xs:element name="poContent" nillable="true" type="xs:string"/>
  </xs:sequence> </xs:complexType> </xs:element>
 <xs:element name="checkResponse"> <xs:complexType> <xs:sequence>
  <xs:element name="return" nillable="true" type="xs:boolean"/>
  </xs:sequence> </xs:complexType> </xs:element>
 </xs:schema>
</wsdl:types>
<wsdl:message name="checkMessage">
 <wsdl:part name="part1" element="ns0:check"/></wsdl:message>
 <wsdl:message name="checkResponse">
  <wsdl:part name="part1" element="ns0:checkResponse"/></wsdl:message>
<wsdl:portType name="VampireInvokerPortType">
 <wsdl:operation name="check">
    <wsdl:input xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
        message="axis2:checkMessage" wsaw:Action="urn:check"/>
   <wsdl:output message="axis2:checkResponse"/>
  </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="VampireInvokerSOAP11Binding"
    type="axis2:VampireInvokerPortType">
 <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
 <wsdl:operation name="check">
    <soap:operation soapAction="urn:check" style="document"/>
    <wsdl:input><soap:body use="literal"/></wsdl:input>
    <wsdl:output><soap:body use="literal"/></wsdl:output>
 </wsdl:operation>
</wsdl:binding>
<wsdl:binding
    name="VampireInvokerSOAP12Binding"
    type="axis2:VampireInvokerPortType">
 <soap12:binding transport="http://schemas.xmlsoap.org/soap/http"
    style="document"/>
<wsdl:operation name="check">
 <soap12:operation soapAction="urn:check" style="document"/>
  <wsdl:input> <soap12:body use="literal"/> </wsdl:input>
   <wsdl:output> <soap12:body use="literal"/> </wsdl:output>
 </wsdl:operation>
</wsdl:binding>
<wsdl:binding
```

```
      name="VampireInvokerHttpBinding"
      type="axis2:VampireInvokerPortType">
   <http:binding verb="POST"/>
   <wsdl:operation name="check">
      <http:operation location="check"/>
      <wsdl:input><mime:content type="text/xml"/> </wsdl:input>
      <wsdl:output><mime:content type="text/xml"/> </wsdl:output>
   </wsdl:operation>
</wsdl:binding>
<wsdl:service name="VampireInvoker">
 <wsdl:port name="VampireInvokerSOAP11port_http"
      binding="axis2:VampireInvokerSOAP11Binding">
    <soap:address location="http://138.232.65.138:8080/axis2/services/VampireInvoker"/>
 </wsdl:port>
 <wsdl:port name="VampireInvokerSOAP12port_http"
      binding="axis2:VampireInvokerSOAP12Binding">
    <soap12:address location="http://138.232.65.138:8080/axis2/services/VampireInvoker"/>
  </wsdl:port>
    <wsdl:port name="VampireInvokerHttpport1"
      binding="axis2:VampireInvokerHttpBinding">
        <http:address location="http://138.232.65.138:8080/axis2/rest/VampireInvoker"/>
   </wsdl:port>
</wsdl:service>
</wsdl:definitions>
```

Listing 4: WSDL for VampireInvoker Web Service

B Domain Ontologies for SWSC Shipment Scenario (WSML)

```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-full"
namespace {
_"http :// members.deri.org/~michaels/sdc/swsc-shipment/ontologies/location.wsml#",
  dc _"http :// purl.org/dc/elements/1.1#",
  loc _"http :// cvs.deri.org/cgi-bin/viewcvs.cgi/*checkout*/wsmo/papers/swsc/Location.wsml#",
  wsml _"http :// www.wsmo.org/wsml/wsml-syntax#" }

ontology
_"http :// members.deri.org/~michaels/sdc/swsc-shipment/ontologies/location.wsml"

  nonFunctionalProperties
     dc#title  hasValue "Location Ontology"
    dc#language hasValue "en-US"
    dc#contributor  hasValue {"Holger Lausen", "Adina Sirbu","Michael Stollberg"}
    dc#format hasValue "text/plain"
    dc#date hasValue _date(2007,03,08)
    wsml#version hasValue "$Version$"
  endNonFunctionalProperties

/*
 * concepts
 */
concept Location
  nfp
    dc#description hasValue "A Location is a place defined by a longitude, latitude ,
        and an altitude. If the altitude is missing, then we assume that it is ground level."
  endnfp
  locatedIn ofType GeographicArea

concept GeographicArea subConceptOf Location
  nfp
    dc#description hasValue "A GeographicArea is a larger area such as a state or country."
  endnfp
  name ofType _string
  containsLocation ofType Location

concept Continent subConceptOf GeographicArea

concept Region subConceptOf GeographicArea

concept Country subConceptOf GeographicArea

concept State subConceptOf GeographicArea

concept WaterArea subConceptOf GeographicArea

concept City subConceptOf GeographicArea

/*
 * axioms
 */

axiom transitivityLocatedIn
  nfp
      dc#description hasValue "defines the transitivity of the locatedIn attribute "
  endnfp
  definedBy
    forAll (?L1,?L2,?L2). ?L1 memberOf Location and ?L2 memberOf Location and ?L3 memberOf Location
      and ?L1[locatedIn hasValue ?L2] and ?L2[locatedIn hasValue ?L3] implies
      ?L1[locatedIn hasValue ?L3].
```

```
axiom locationsInContinentsAreDistinct
  nfp
      dc#description hasValue "Locations in different continents are disjoint"
  endnfp
  definedBy
   forAll (?C1,?C2,?L). ?C1 memberOf continent and ?C2 memberOf continent and
     ?L[locatedIn hasValue ?C1] implies neg ?L[locatedIn hasValue ?C2].


axiom locationsInCountriesAreDistinct
  nfp
      dc#description hasValue "Locations in different continents are disjoint"
  endnfp
  definedBy
   forAll (?C1,?C2,?L). ?C1 memberOf country and ?C2 memberOf country and
     ?L[locatedIn hasValue ?C1] implies neg ?L[locatedIn hasValue ?C2].


/*
 * continents and regions
 */

instance World memberOf Location

instance Africa memberOf Continent
  name hasValue "Africa"
  locatedIn hasValue World

instance Antarctica memberOf Continent
  name hasValue "Antarctica"
  locatedIn hasValue World

instance Asia memberOf Continent
  name hasValue "Asia"
  locatedIn hasValue World

instance Australia memberOf Continent
  name hasValue "Australia"
  locatedIn hasValue World

instance Europe memberOf Continent
  name hasValue "Europe"
  locatedIn hasValue World

instance NorthAmerica memberOf Continent
  name hasValue "North America"
  locatedIn hasValue World

instance SouthAmerica memberOf Continent
  name hasValue "South America"
  locatedIn hasValue World

instance Oceania memberOf Region
  name hasValue "Oceania"
  locatedIn hasValue World
  containsLocation hasValue Australia


/*
 * countries
 */
instance AF memberOf Country
  name hasValue "Afghanistan"
  locatedIn hasValue Asia
```

```
instance AS memberOf Country
  name hasValue "American Samoa"
  locatedIn hasValue Oceania

// we omit the listing of all countries with respect to space

/*
 * cities
 */
instance Bristol memberOf City
  name hasValue "Bristol"
  locatedIn hasValue GB

instance NewYork memberOf City
  name hasValue "New York"
  locatedIn hasValue US

instance Tunis memberOf City
  name hasValue "Tunis"
  locatedIn hasValue TN

instance Luxembourg memberOf City
  name hasValue "Luxembourg"
  locatedIn hasValue LU
```

Listing 5: Location Ontology

```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-full"
namespace {
_"http :// members.deri.org/~michaels/sdc/swsc-shipment/ontologies/shipment.wsml#",
  dc _"http :// purl.org/dc/elements/1.1#",
  loc _"http :// cvs.deri.org/cgi-bin/viewcvs.cgi/*checkout*/wsmo/papers/swsc/Location.wsml#",
  wsml _"http :// www.wsmo.org/wsml/wsml-syntax#" }

ontology
_"http :// members.deri.org/~michaels/sdc/swsc-shipment/ontologies/shipment.wsml"

importsOntology
{_"http :// members.deri.org/~michaels/sdc/swsc-shipment/ontologies/location.wsml"}

  nonFunctionalProperties
    dc#title hasValue "Shipment Ontology"
    dc#language hasValue "en-US"
    dc#contributor hasValue {"Michael Stollberg"}
    dc#format hasValue "text/plain"
    dc#date hasValue _date(2007,03,08)
    wsml#version hasValue "Version"
  endNonFunctionalProperties

/*
 * concepts
 */

concept shipmentOrder
 from impliesType Sender
 to impliesType Receiver
 item impliesType Package
 price impliesType Price

concept Sender
 address impliesType loc#Location
```

```
concept Receiver
 address impliesType loc#Location

concept Price
 amount impliesType wsml#float
 currency impliesType Currency

concept Currency
 name impliesType wsml#string
 symbol impliesType wsml#string


concept Package
 weight impliesType WeightClass
 size  impliesType Size


concept WeightClass
 nfp
    dc#description hasValue "weights are distinguished n 10 lbs classes.
    This is a sufficient abstraction for the intended usage."
 endnfp
 includedIn impliesType WeightClass

axiom transitiveWeightClassInclusion
 nfp
    dc#description hasValue "defines the relationship of weight classes"
 endnfp
 definedBy
    forAll (?W1,?W2,?W3). ?W1 memberOf WeightClass and ?W2 memberOf WeightClass
     and ?W3 memberOf WeightClass
     and ?W1[includedIn hasValue ?W2] and ?W2[includedIn hasValue ?W3]
     implies ?W1[includedIn hasValue ?W3].

/*
 * instances
 */

instance USD memberOf Currency
 name hasValue "US Dollar"
 symbol hasValue "USD"


instance heavy memberOf WeightClass
 includedIn hasValue heavy
instance w70lq memberOf WeightClass
 includedIn hasValue heavy
instance w60lq memberOf WeightClass
 includedIn hasValue w70lq
instance w50lq memberOf WeightClass
 includedIn hasValue w60lq
instance w40lq memberOf WeightClass
 includedIn hasValue w50lq
instance w30lq memberOf WeightClass
 includedIn hasValue w40lq
instance w20lq memberOf WeightClass
 includedIn hasValue w30lq
instance light memberOf WeightClass
 includedIn hasValue w20lq
 includedIn hasValue light
```

Listing 6: Shipment Ontology

C Evaluation Data

This appendix provides the statistical prepared data of the the SDC evaluation results presented in Section 3.2.2. The original data along with logs of the comparison test run are available for download on the SDC homepage (see `http://members.deri.at/~michaels/software/sdc/`).

For a proper evaluation and discussion, we apply statistical standard notions to the comparison data. Table 4 provides an overview of the used notions (in accordance to [11]).

Table 4: Overview of Used Statistical Notions

Notion	**Description**	**Formula***
arithmetic mean μ	denotes the middle value of a data set	$\mu = \frac{1}{n} \times \sum_{i=1}^{n} x_i$
median $\bar{x}$	denotes the middle value v_m of a data set such that 50 % of the values are smaller and 50 % are bigger than v_m; in contrast to μ, here v_m is not determined by extreme values	if n is even: $\bar{x} = \frac{x_{i-1}+x_i}{2}$ with $i = \frac{n}{2} + 1$
quartile	divides a data set into 3 parts: (1) $\bar{x}_{.25}$: 25% are smaller and 75% are bigger than this value, (2) $\bar{x}_{.5} = \bar{x}$, and (3) $\bar{x}_{.75}$: 75% are smaller and 25% are bigger than this value	$\bar{x}_{.25} = x_i$ with $i = \downarrow 0.25 \times \frac{n+1}{2}$ $\bar{x}_{.75} = x_i$ with $i = \downarrow 0.75 \times \frac{n+1}{2}$
variance σ^2	a measurement of the statistical dispersion of a data set	$\sigma^2 = \frac{1}{n} \times \sum_{i=1}^{n} (x_i - \mu)^2$
standard deviation	a measure of the spread of the values in a data set, defined as the square root of the variance	$\sigma = \sqrt{\frac{1}{n} \times \sum_{i=1}^{n} (x_i - \mu)^2}$
coefficient of variation	the relation between the standard deviation σ and the arithmetic mean μ, a measurement for the value dispersion that allows to compare data sets with very different value ranges	$\frac{\sigma}{\mu}$

symbols: x_i = value of a data item at position i; n = number of data items (here: entire population)

In the SWSC shipment scenario – with the resource modelling described in Section 3.1.2) – a single matchmaking operation takes in average 115 milliseconds (max. 15 ms for the Java part, ca. 20 ms for the Web service invocation, and the rest for the matchmaking via proof; the latter aspect strongly depends on the complexity of the functional descriptions which, in this comparison, is relatively low).

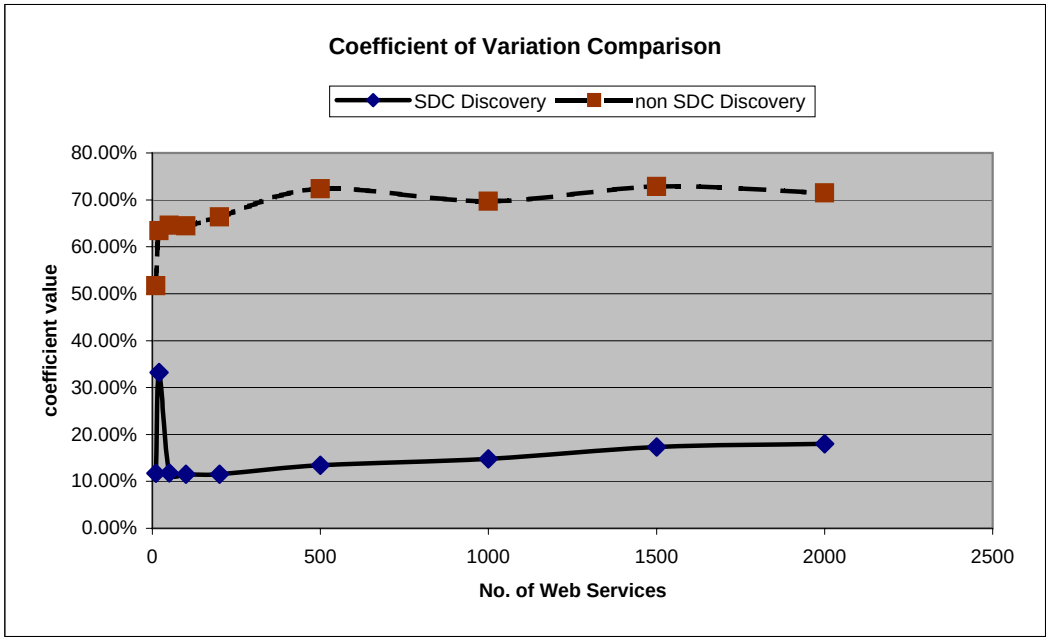
statistical preparation of comparison for all 10 goal instances
data = arithmetic mean of the data for the individual goal instances (see following worksheets)

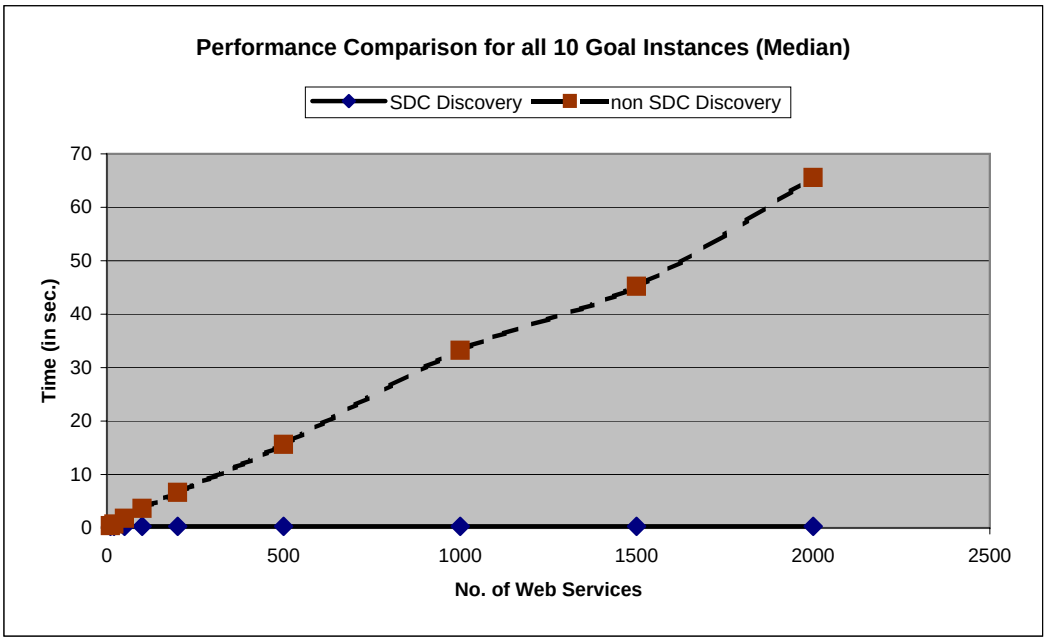
SDC Discovery

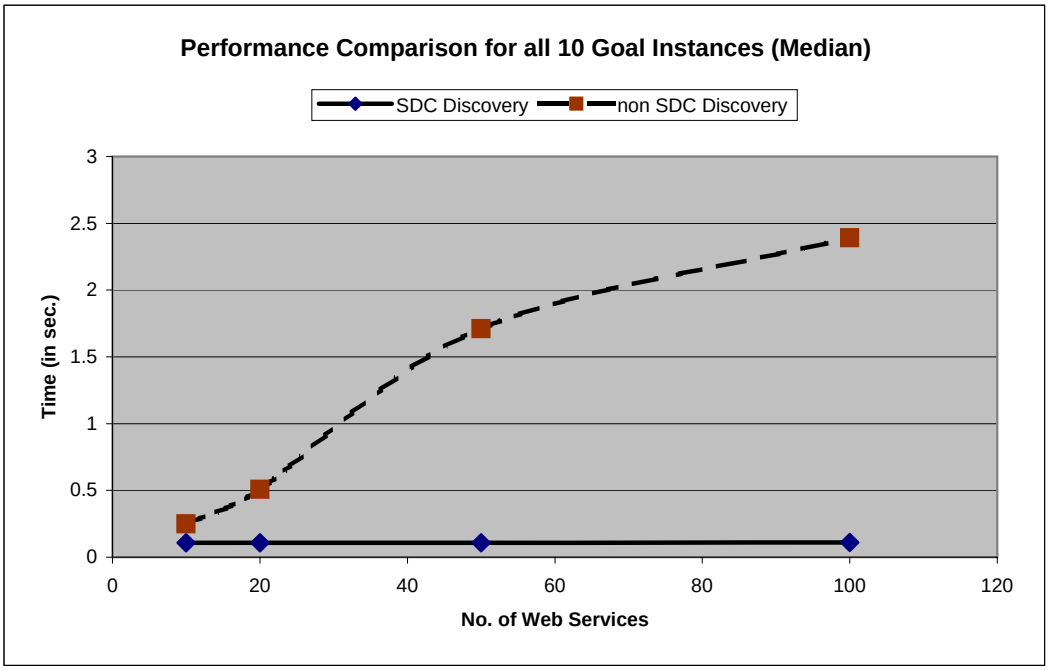
no. WS	arith. mean (in sec)	median (in sec)	min (in sec)	max (in sec)	variance (in msec^2)	stand. deviation (in sec)	coefficient of variation
10	0.28116543	0.2734	0.2559	0.4563	1562.564515	0.032997502	11.74%
20	0.29307	0.27885	0.2574	0.9346	48644.8878	0.097271044	33.19%
50	0.284648	0.27885	0.2575	0.4595	1568.8552	0.033559762	11.79%
100	0.287544	0.28125	0.2651	0.4533	1489.18936	0.03314478	11.53%
200	0.290118	0.2815	0.2636	0.4359	1759.13332	0.033389555	11.51%
500	0.29223	0.2859	0.2652	0.4922	2384.44284	0.039230557	13.42%
1000	0.296686	0.2875	0.2651	0.539	2795.6794	0.043867008	14.79%
1500	0.301862	0.2906	0.2717	0.594	4701.46044	0.052346692	17.34%
2000	0.306008	0.2938	0.2731	0.5954	4376.71728	0.055161952	18.03%

Non-SDC Discovery

no. WS	arith. mean (in sec)	median (in sec)	min (in sec)	max (in sec)	variance (in msec^2)	stand.deviation (in sec)	coefficient of variation
10	0.413052	0.3922	0.1544	0.9438	49210.50728	0.213581785	51.71%
20	0.815156	0.733	0.159	2.3708	292723.8776	0.517229873	63.45%
50	2.000464	1.79795	0.1888	5.054	1759016.815	1.292189794	64.59%
100	3.962612	3.67855	0.253	10.4883	7027879.395	2.554986046	64.48%
200	7.60848	6.6697	0.3484	19.5459	26774771.26	5.047951866	66.35%
500	18.334298	15.6115	0.7014	51.321	186903202.9	13.26301462	72.34%
1000	37.696672	33.2215	2.1126	103.743	725864586	26.27576623	69.70%
1500	53.90973	45.1997	1.4599	148.776	1632655033	39.26855982	72.84%
2000	72.963554	65.5562	2.3656	192.97	2899274797	52.13448486	71.45%




Performance Comparison for all 10 Goal Instances (Median)
SDC Discovery
non SDC Discovery
Time (in sec.)
70
60
50
40
30
20
10
0
0
500
1000
1500
2000
2500
No. of Web Services


Performance Comparison for all 10 Goal Instances (Median)
SDC Discovery
non SDC Discovery
Time (in sec.)
3
2.5
2
1.5
1
0.5
0
0
20
40
60
80
100
120
No. of Web Services

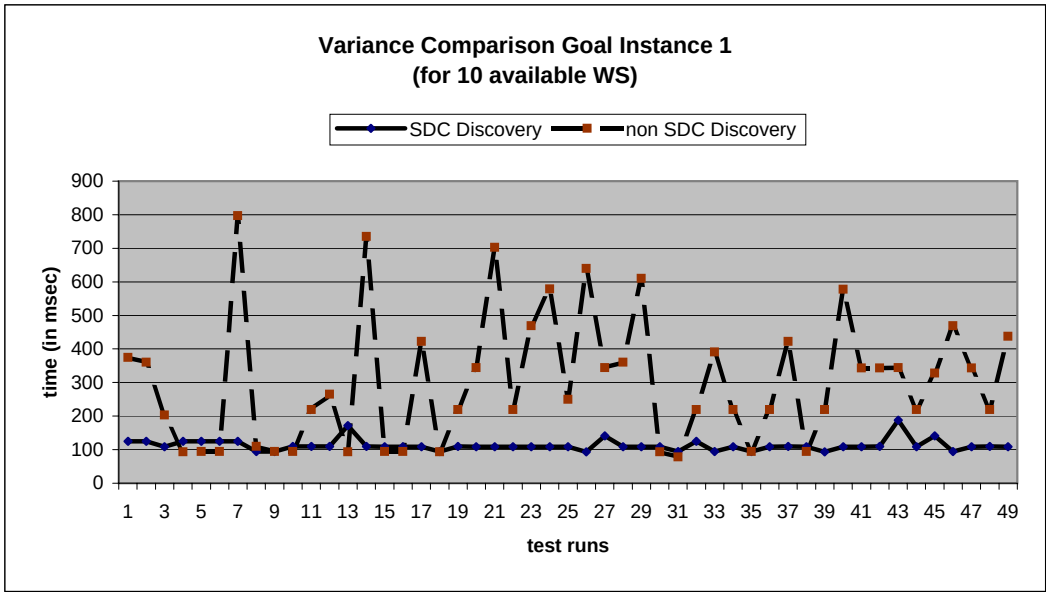
statistical preparation of comparison for: goal instance 1

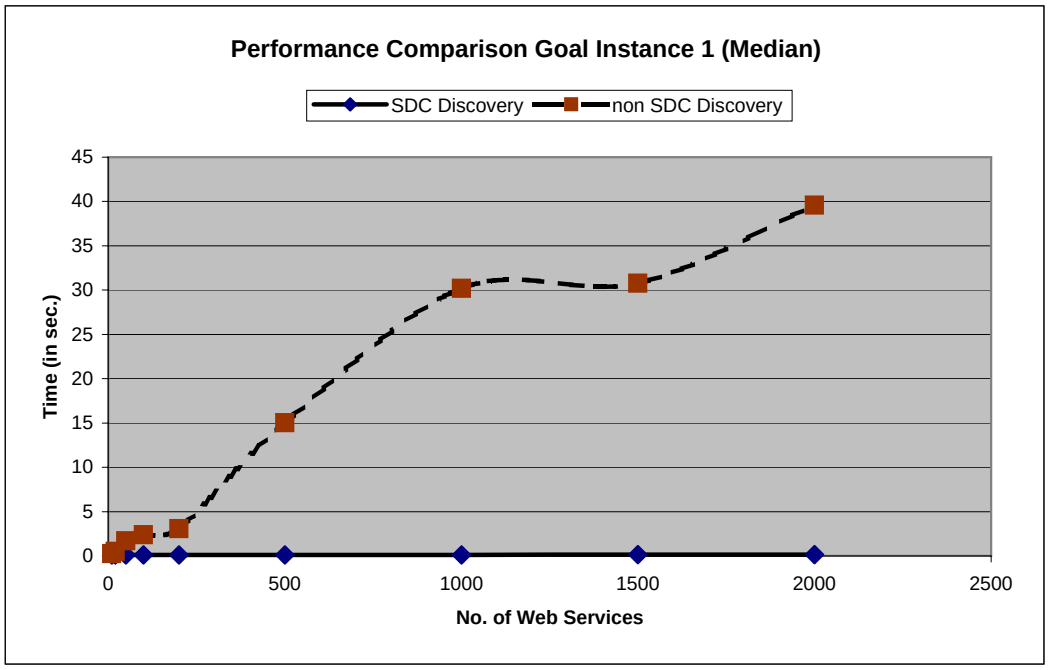
SDC Discovery

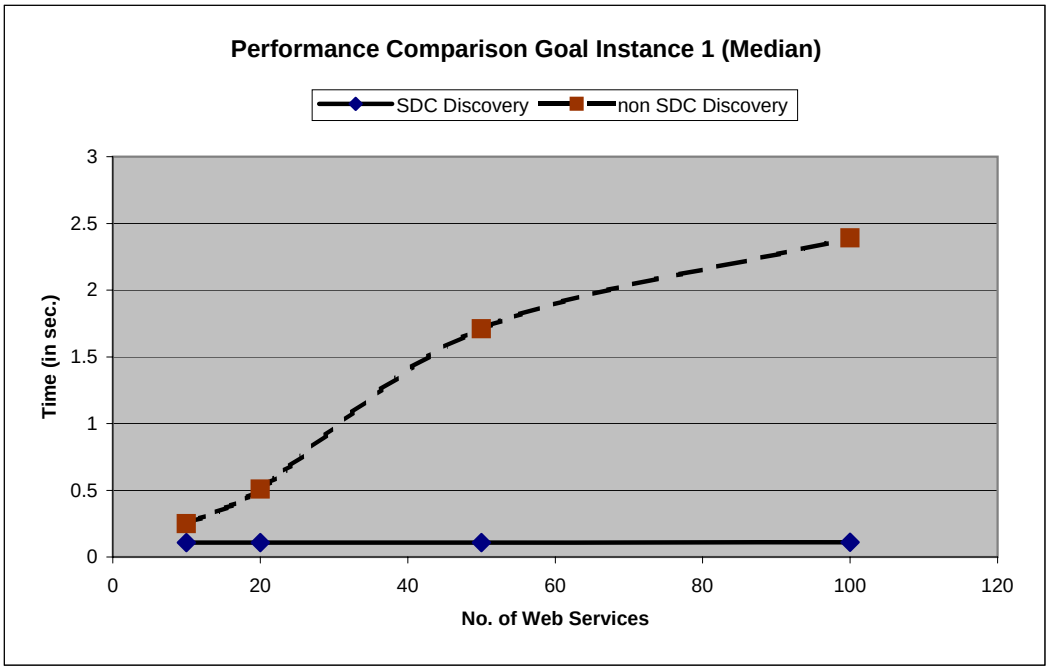
no. WS	arith. mean (in sec)	median (in sec)	min (in sec)	max (in sec)	variance (in msec^2)	stand. deviation (in sec)	coefficient of variation
10	0.11285714	0.109	0.093	0.188	314.6122449	0.017737312	15.72%
20	0.1115	0.109	0.093	0.203	323.13	0.017975817	16.12%
50	0.11246	0.109	0.093	0.203	234.2084	0.015303869	13.61%
100	0.11616	0.11	0.109	0.156	118.5744	0.010889187	9.37%
200	0.125	0.11	0.109	0.219	803.72	0.028349956	22.68%
500	0.12626	0.125	0.109	0.219	478.1124	0.021865781	17.32%
1000	0.12966	0.125	0.109	0.234	274.8644	0.016579035	12.79%
1500	0.14556	0.14	0.125	0.297	935.6464	0.030588338	21.01%
2000	0.15252	0.141	0.125	0.453	3213.8096	0.056690472	37.17%

Non-SDC Discovery

no. WS	arith. mean (in sec)	median (in sec)	min (in sec)	max (in sec)	variance (in msec^2)	stand.deviation (in sec)	coefficient of variation
10	0.29976	0.25	0.078	0.797	35511.9024	0.18844602	62.87%
20	0.6147	0.508	0.078	1.938	208179.49	0.456266907	74.23%
50	1.77864	1.7105	0.093	4.031	1384656.55	1.176714303	66.16%
100	3.0873	2.391	0.094	8.593	5279921.09	2.297807888	74.43%
200	5.64736	3.0545	0.078	19.608	27426398.71	5.237021931	92.73%
500	17.12608	15.0305	0.094	46.044	146719239.3	12.11277174	70.73%
1000	31.25436	30.1855	0.344	92.165	566908810.1	23.80984691	76.18%
1500	45.28732	30.7865	0.828	147.067	1445426988	38.01877153	83.95%
2000	56.24786	39.583	1.422	175.487	2542395428	50.42217199	89.64%




Performance Comparison Goal Instance 1 (Median)
SDC Discovery
non SDC Discovery
Time (in sec.)
45
40
35
30
25
20
15
10
5
0
0
500
1000
1500
2000
2500
No. of Web Services


Performance Comparison Goal Instance 1 (Median)
SDC Discovery
non SDC Discovery
Time (in sec.)
3
2.5
2
1.5
1
0.5
0
0
20
40
60
80
100
120
No. of Web Services

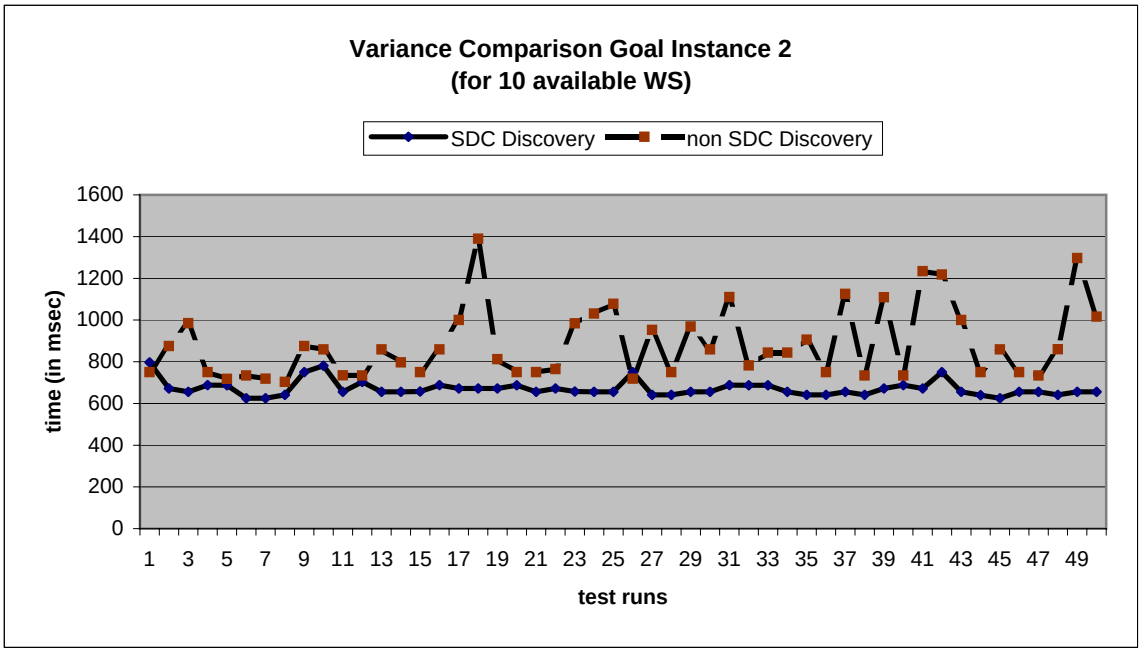
statistical preparation of comparison for: goal instance 2

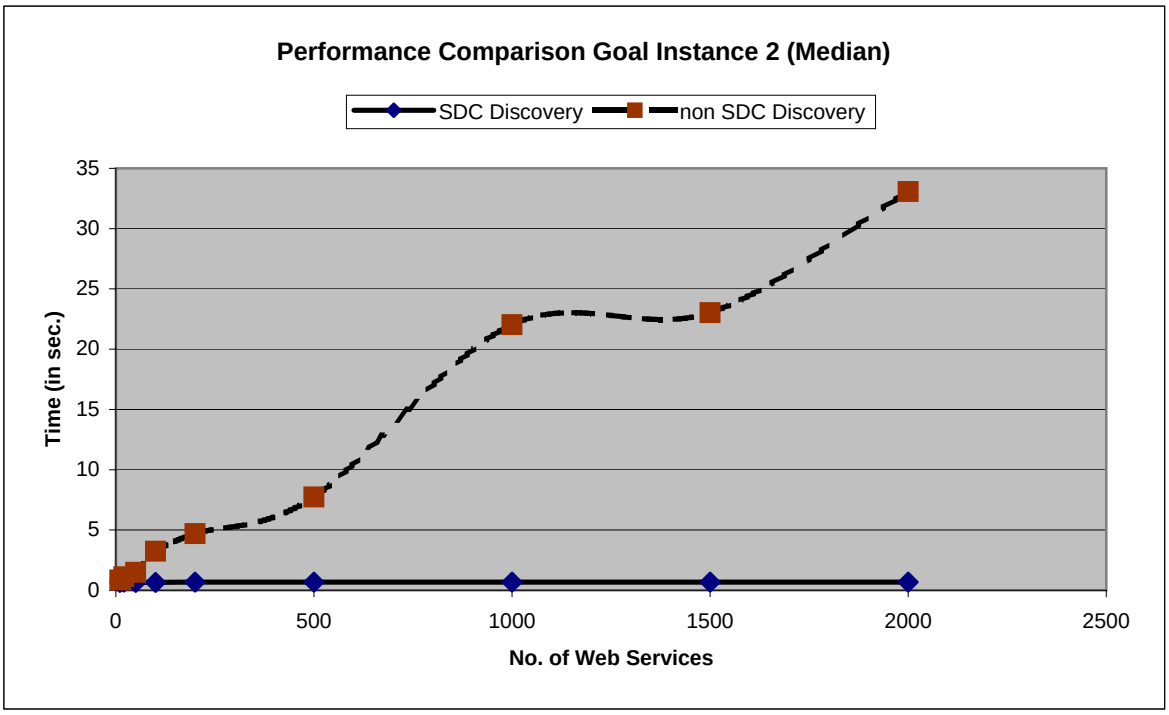
SDC Discovery

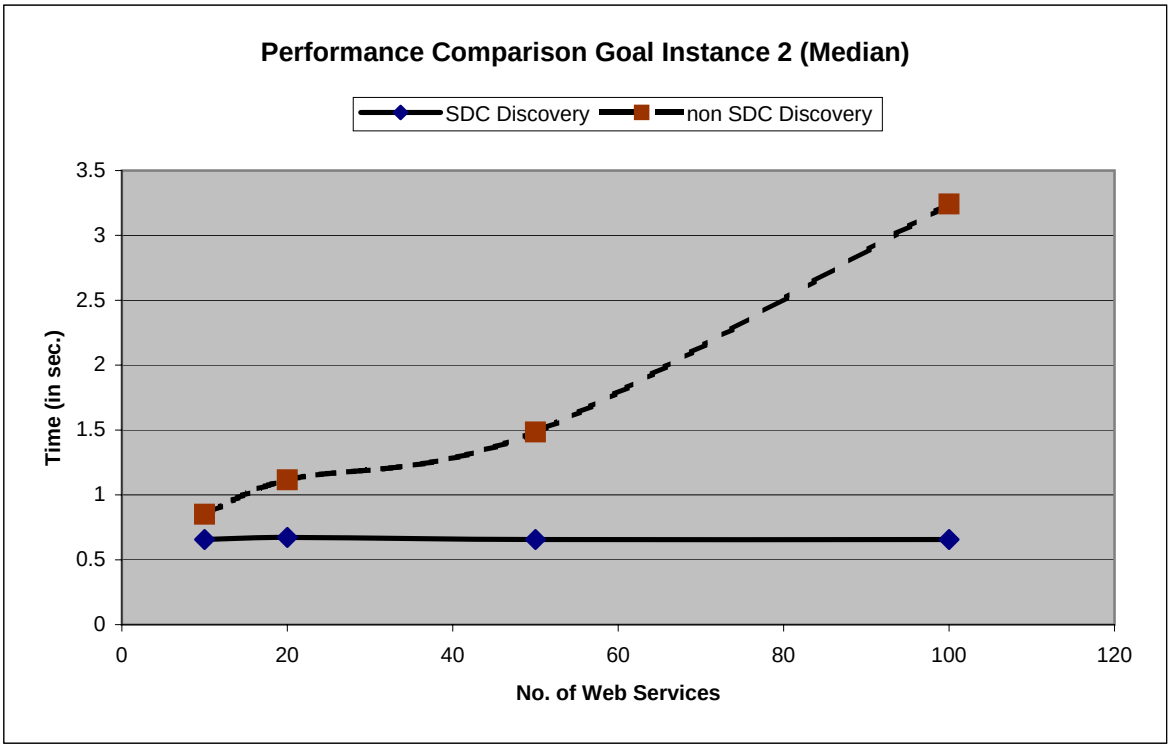
no. WS	arith. mean (in sec)	median (in sec)	min (in sec)	max (in sec)	variance (in msec^2)	stand. deviation (in sec)	coefficient of variation
10	0.67094	0.656	0.625	0.781	1046.844648	0.032354979	4.82%
20	0.68506	0.672	0.625	1.047	4271.8564	0.06535944	9.54%
50	0.67156	0.656	0.625	0.797	1659.7264	0.04073974	6.07%
100	0.67366	0.6565	0.64	0.797	1177.7044	0.034317698	5.09%
200	0.68524	0.672	0.64	0.828	1599.3024	0.039991279	5.84%
500	0.67462	0.672	0.64	0.766	816.8756	0.028581036	4.24%
1000	0.6834	0.672	0.624	0.844	1852.08	0.043035799	6.30%
1500	0.70454	0.672	0.641	1.407	14129.2884	0.118866683	16.87%
2000	0.68118	0.672	0.64	0.797	1156.9076	0.034013344	4.99%

Non-SDC Discovery

no. WS	arith. mean (in sec)	median (in sec)	min (in sec)	max (in sec)	variance (in msec^2)	stand.deviation (in sec)	coefficient of variation
10	0.88336	0.8515	0.703	1.39	28242.5504	0.1680552	19.02%
20	1.18666	1.1165	0.718	2.266	136750.5044	0.369797924	31.16%
50	1.76084	1.485	0.734	3.937	732013.8144	0.855578059	48.59%
100	3.38882	3.242	0.75	8.156	3757901.788	1.938530832	57.20%
200	5.02378	4.703	0.766	12.717	10643375.65	3.262418681	64.94%
500	10.48922	7.734	0.75	33.014	69717564.21	8.349704439	79.60%
1000	28.59172	22.0375	0.828	106.57	659784393.8	25.68626858	89.84%
1500	33.92656	23.023	2.052	107.751	781996608.8	27.96420227	82.43%
2000	51.03844	33.076	2.031	162.254	1768918426	42.05851193	82.41%




Performance Comparison Goal Instance 2 (Median)
SDC Discovery
non SDC Discovery
Time (in sec.)
0
5
10
15
20
25
30
35
0
500
1000
1500
2000
2500
No. of Web Services


Performance Comparison Goal Instance 2 (Median)
SDC Discovery
non SDC Discovery
Time (in sec.)
0
0.5
1
1.5
2
2.5
3
3.5
0
20
40
60
80
100
120
No. of Web Services

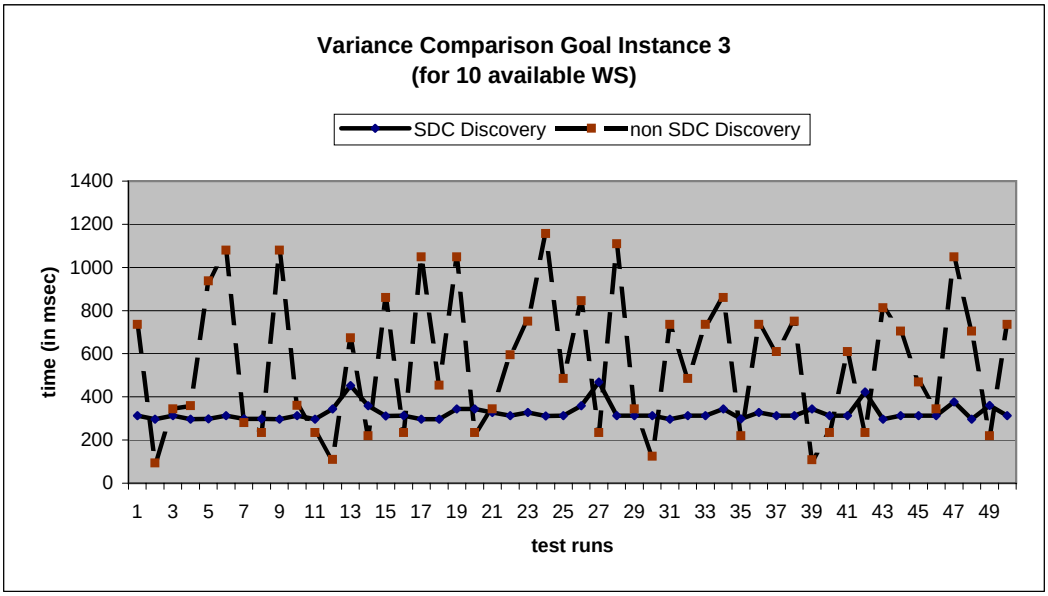
statistical preparation of comparison for: goal instance 3

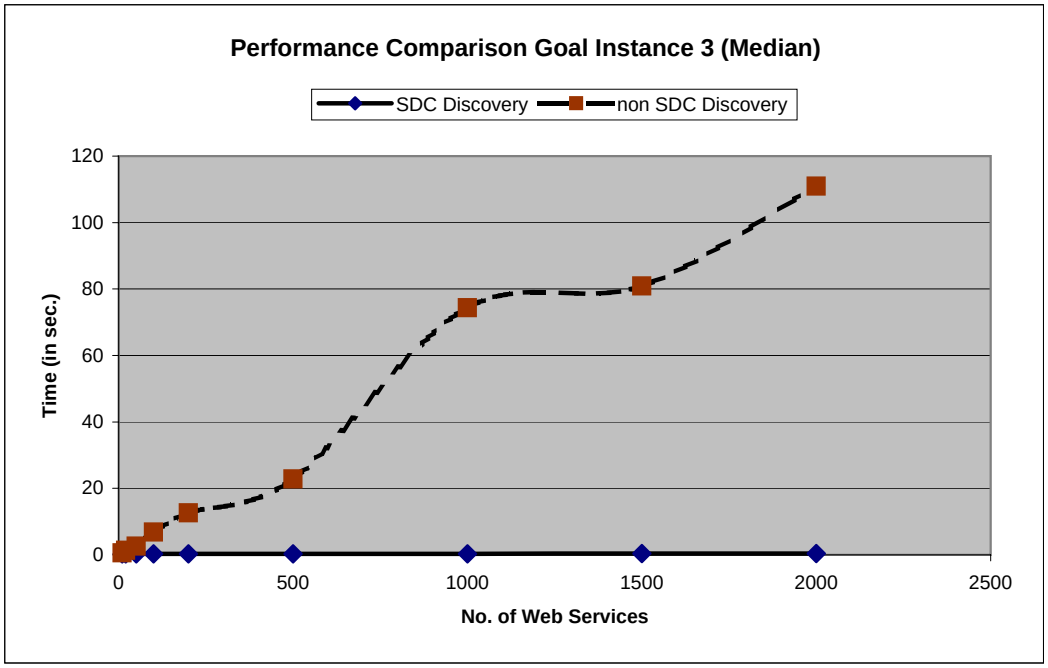
SDC Discovery

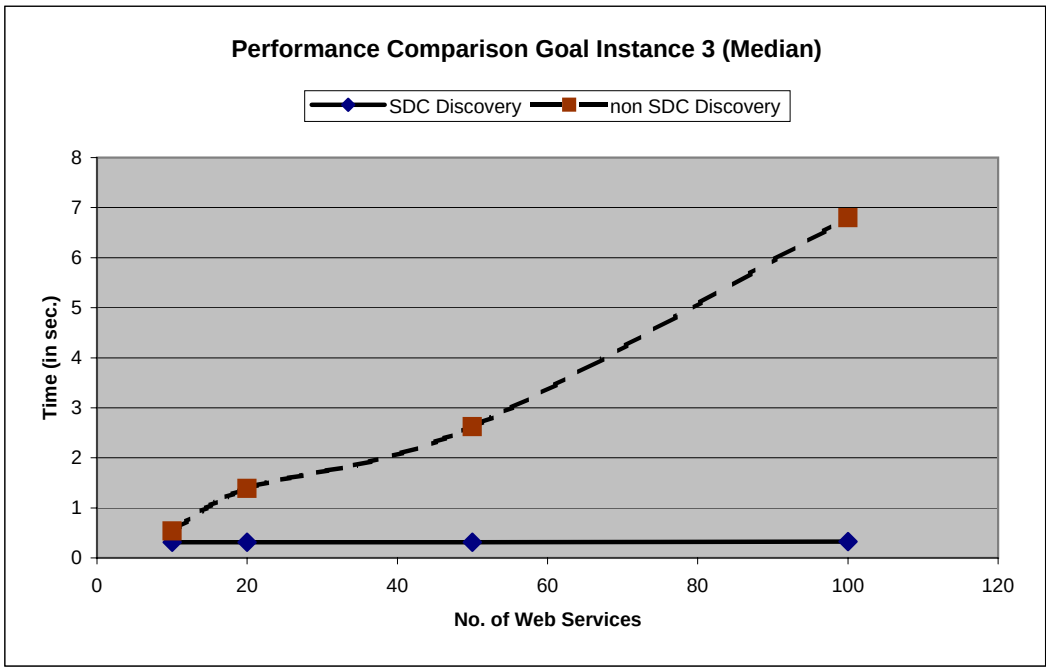
no. WS	arith. mean (in sec)	median (in sec)	min (in sec)	max (in sec)	variance (in msec^2)	stand. deviation (in sec)	coefficient of variation
10	0.32526531	0.313	0.297	0.469	1378.399	0.037126796	11.41%
20	0.31928	0.313	0.297	0.422	478.9616	0.021885191	6.85%
50	0.33648	0.313	0.297	0.688	3580.8096	0.059839866	17.78%
100	0.32798	0.328	0.312	0.422	462.0196	0.021494641	6.55%
200	0.33442	0.328	0.312	0.438	980.2436	0.031308842	9.36%
500	0.3472	0.328	0.312	0.812	4866.6	0.069761021	20.09%
1000	0.34028	0.329	0.312	0.437	410.9616	0.020272188	5.96%
1500	0.35066	0.344	0.328	0.532	1559.2644	0.039487522	11.26%
2000	0.36634	0.344	0.328	0.704	3561.5044	0.059678341	16.29%

Non-SDC Discovery

no. WS	arith. mean (in sec)	median (in sec)	min (in sec)	max (in sec)	variance (in msec^2)	stand.deviation (in sec)	coefficient of variation
10	0.55926	0.5395	0.094	1.157	98894.4724	0.314474915	56.23%
20	1.27992	1.3845	0.109	2.408	448213.9536	0.669487829	52.31%
50	2.72448	2.6195	0.109	5.912	2945607.13	1.716277113	62.99%
100	6.74048	6.799	0.703	12.363	12168460.69	3.488332078	51.75%
200	13.18636	12.649	1.391	26.533	50271689.63	7.090253143	53.77%
500	27.5141	22.812	0.937	75.655	357211973.1	18.9000522	68.69%
1000	71.98234	74.3735	7.203	121.857	1009630366	31.7746812	44.14%
1500	93.33886	80.9435	2.078	181.808	2691333380	51.87806261	55.58%
2000	117.40564	110.901	0.391	244.951	5246537195	72.43298416	61.69%




Performance Comparison Goal Instance 3 (Median)
SDC Discovery
non SDC Discovery
Time (in sec.)
120
100
80
60
40
20
0
0
500
1000
1500
2000
2500
No. of Web Services


Performance Comparison Goal Instance 3 (Median)
SDC Discovery
non SDC Discovery
Time (in sec.)
8
7
6
5
4
3
2
1
0
0
20
40
60
80
100
120
No. of Web Services

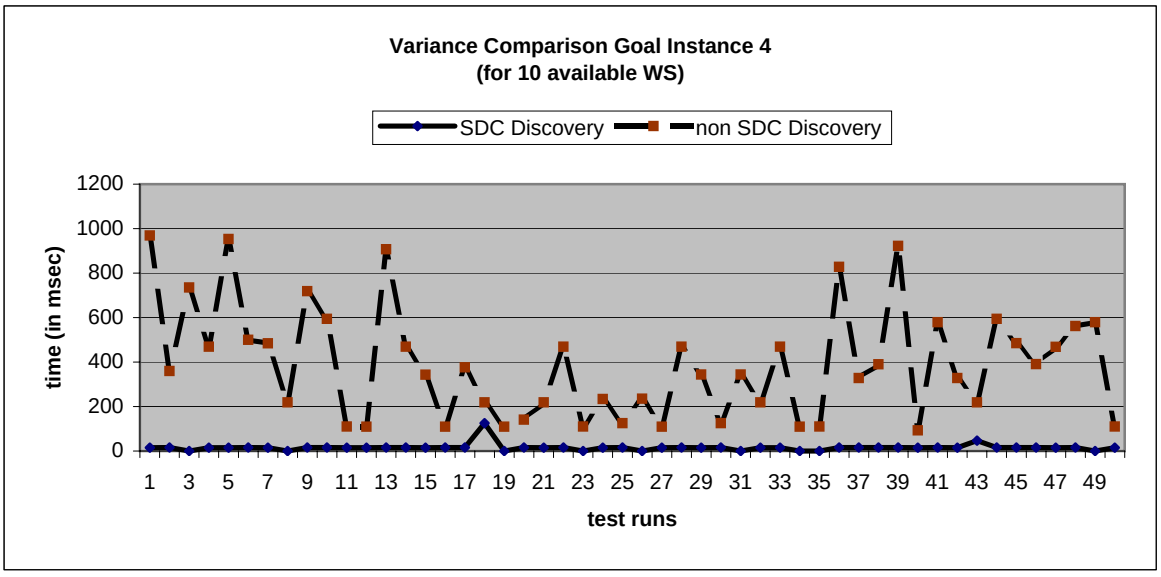
statistical preparation of comparison for: goal instance 4

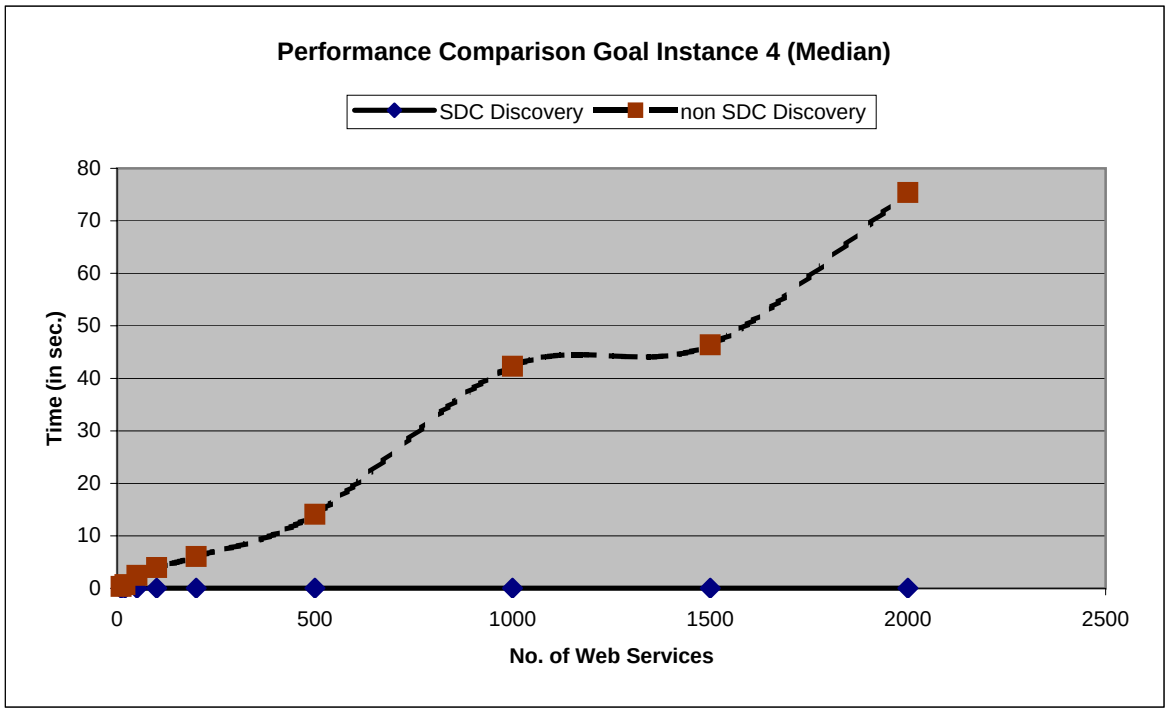
SDC Discovery

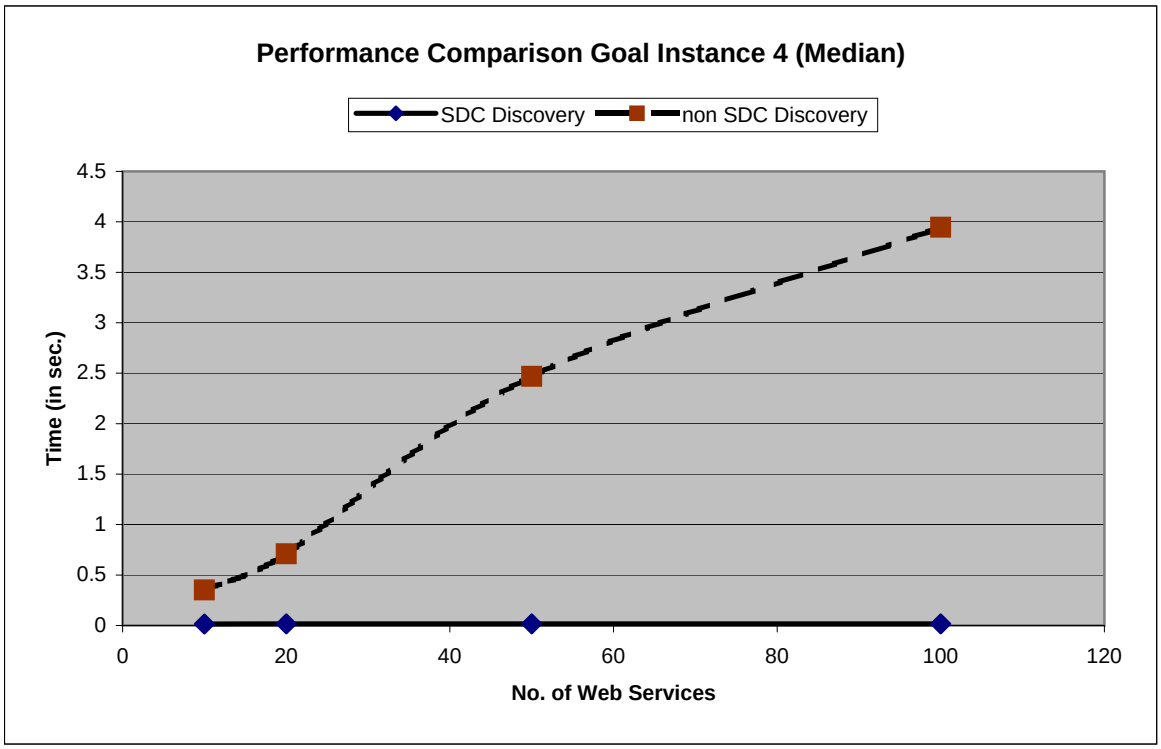
no. WS	arith. mean (in sec)	median (in sec)	min (in sec)	max (in sec)	variance (in msec^2)	stand. deviation (in sec)	coefficient of variation
10	0.01559184	0.015	0	0.125	309.2619742	0.017585846	112.79%
20	0.0162	0.016	0	0.125	273.16	0.016527553	102.02%
50	0.01658	0.016	0	0.125	461.7236	0.021487755	129.60%
100	0.01746	0.016	0	0.125	270.4484	0.016445315	94.19%
200	0.01654	0.016	0	0.047	33.4084	0.00578	34.95%
500	0.01816	0.016	0	0.141	591.1744	0.024314078	133.89%
1000	0.01746	0.0155	0	0.125	514.4084	0.022680573	129.90%
1500	0.01904	0.016	0	0.141	665.5184	0.025797643	135.49%
2000	0.02312	0.016	0	0.328	2362.7456	0.048608082	210.24%

Non-SDC Discovery

no. WS	arith. mean (in sec)	median (in sec)	min (in sec)	max (in sec)	variance (in msec^2)	stand.deviation (in sec)	coefficient of variation
10	0.3892	0.352	0.093	0.969	61639.32	0.248272673	63.79%
20	0.74056	0.711	0.093	1.954	262068.3664	0.511926134	69.13%
50	2.5007	2.4695	0.234	5.689	2555616.17	1.598629466	63.93%
100	4.64586	3.947	0.109	13.333	9679545.68	3.111196824	66.97%
200	7.75394	6.08	0.234	20.49	33590662.5	5.795745206	74.75%
500	18.48602	14.121	0.609	57.283	223885820.5	14.96281459	80.94%
1000	44.49564	42.294	3.361	92.841	713497450.4	26.71137305	60.03%
1500	59.13076	46.3745	0.328	164.794	2319156593	48.15762237	81.44%
2000	77.48424	75.3945	9.187	224.395	2923191810	54.06654982	69.78%




Performance Comparison Goal Instance 4 (Median)
SDC Discovery
non SDC Discovery
Time (in sec.)
0
10
20
30
40
50
60
70
80
0
500
1000
1500
2000
2500
No. of Web Services


Performance Comparison Goal Instance 4 (Median)
SDC Discovery
non SDC Discovery
Time (in sec.)
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
0
20
40
60
80
100
120
No. of Web Services

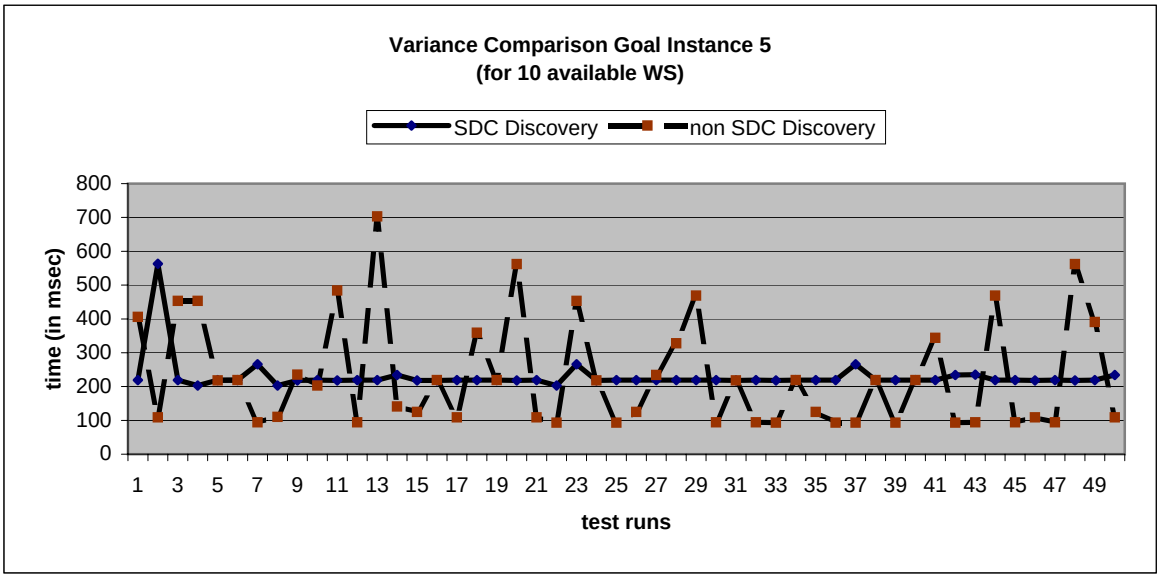
statistical preparation of comparison for: goal instance 5

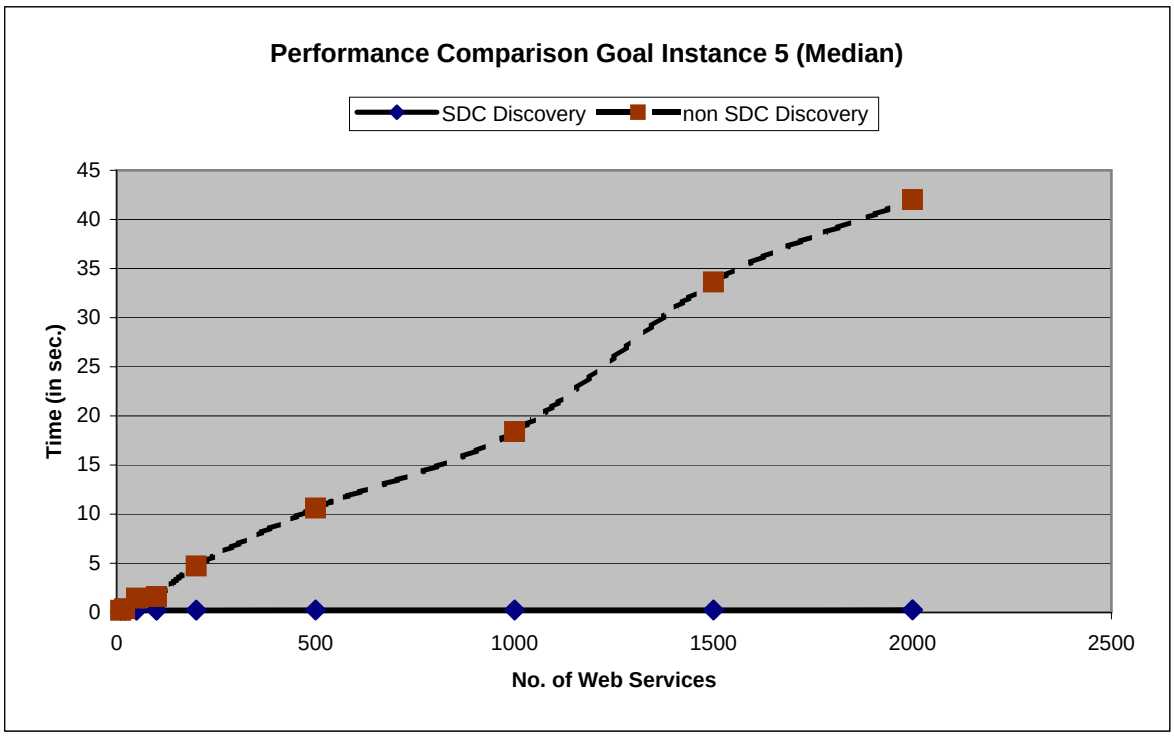
SDC Discovery

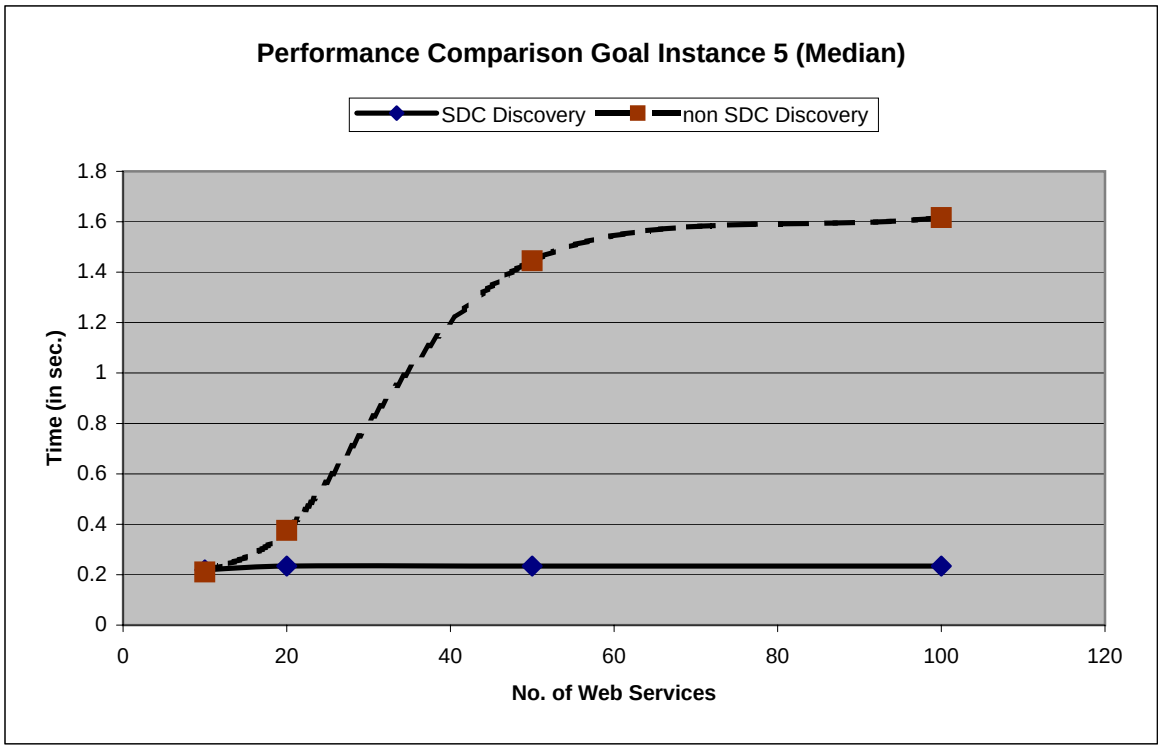
no. WS	arith. mean (in sec)	median (in sec)	min (in sec)	max (in sec)	variance (in msec^2)	stand. deviation (in sec)	coefficient of variation
10	0.22895918	0.219	0.203	0.563	2485.957518	0.049859377	21.78%
20	0.23834	0.234	0.218	0.641	3404.5844	0.058348817	24.48%
50	0.2428	0.234	0.218	0.594	3278.76	0.057260458	23.58%
100	0.24152	0.234	0.218	0.36	607.6896	0.024651361	10.21%
200	0.24244	0.235	0.218	0.281	169.0064	0.013000246	5.36%
500	0.25106	0.25	0.234	0.36	560.6564	0.023678184	9.43%
1000	0.25108	0.25	0.234	0.312	198.1936	0.014078125	5.61%
1500	0.25904	0.25	0.25	0.328	243.7184	0.015611483	6.03%
2000	0.26252	0.265	0.25	0.344	284.5296	0.016868005	6.43%

Non-SDC Discovery

no. WS	arith. mean (in sec)	median (in sec)	min (in sec)	max (in sec)	variance (in msec^2)	stand.deviation (in sec)	coefficient of variation
10	0.23112	0.2105	0.093	0.703	25191.6256	0.1587187	68.67%
20	0.47244	0.3755	0.093	1.422	120317.8064	0.346868572	73.42%
50	1.49648	1.445	0.094	4.422	963954.8896	0.981812044	65.61%
100	1.91464	1.6165	0.093	7.468	2735509.83	1.653937674	86.38%
200	5.52624	4.726	0.094	21.139	20944864.22	4.576555935	82.82%
500	11.78104	10.6245	0.218	39.769	88498622.12	9.407370627	79.85%
1000	23.13188	18.397	1.688	73.95	300887400.4	17.3461062	74.99%
1500	40.1553	33.6255	0.453	144.159	1247036569	35.31340495	87.94%
2000	46.35394	42.0245	1.39	143.83	1493547346	38.64644027	83.37%




Performance Comparison Goal Instance 5 (Median)
SDC Discovery
non SDC Discovery
Time (in sec.)
0
5
10
15
20
25
30
35
40
45
0
500
1000
1500
2000
2500
No. of Web Services


Performance Comparison Goal Instance 5 (Median)
SDC Discovery
non SDC Discovery
Time (in sec.)
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
0
20
40
60
80
100
120
No. of Web Services

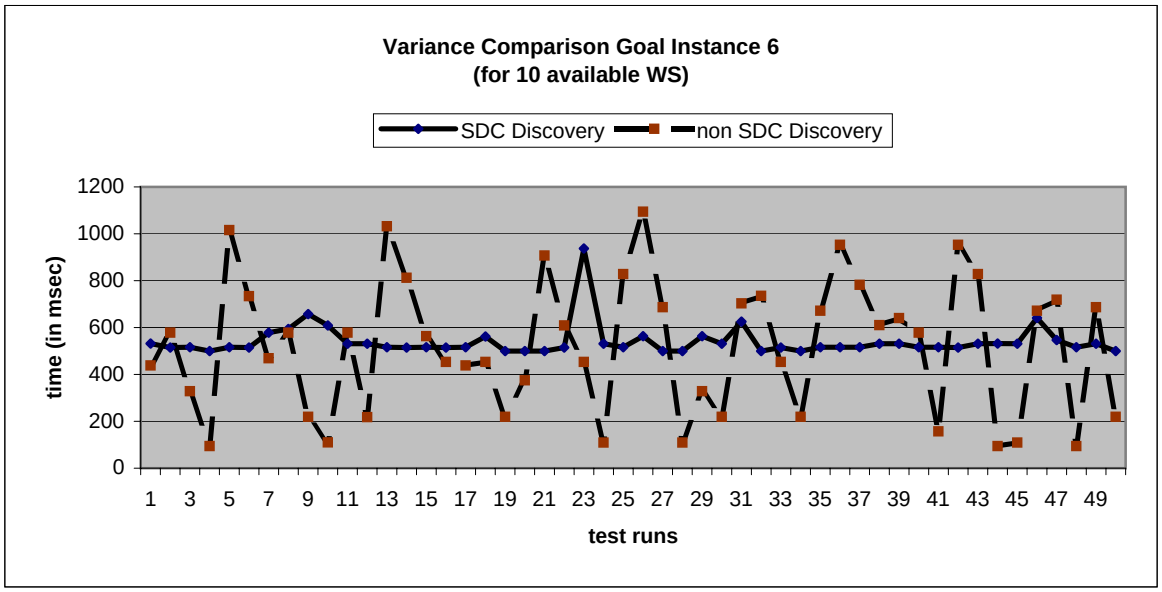
statistical preparation of comparison for: goal instance 6

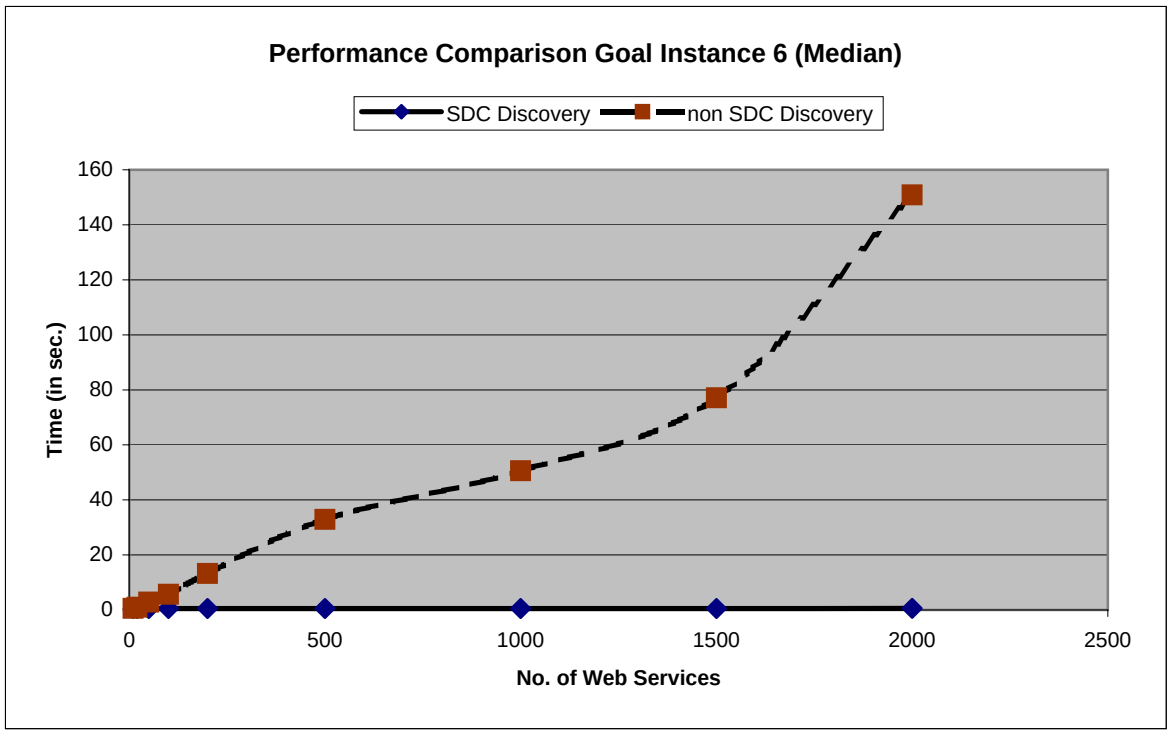
SDC Discovery

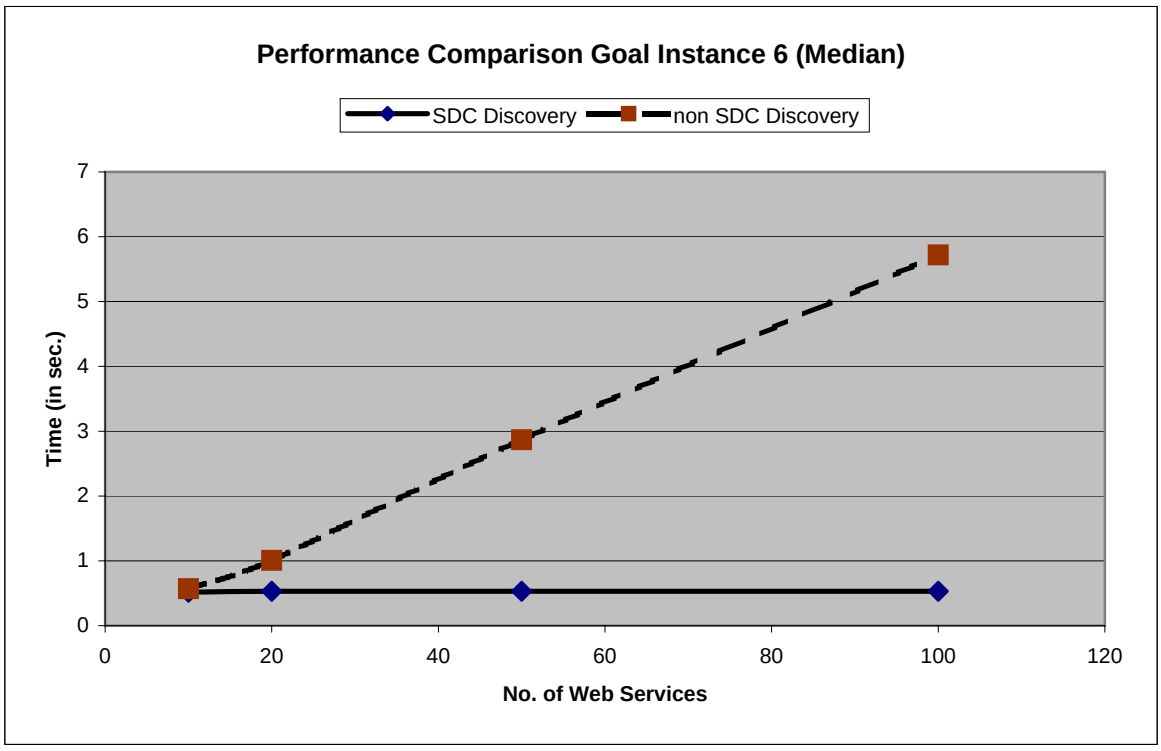
no. WS	arith. mean (in sec)	median (in sec)	min (in sec)	max (in sec)	variance (in msec^2)	stand. deviation (in sec)	coefficient of variation
10	0.5404898	0.516	0.5	0.937	4610.290712	0.067899122	12.56%
20	0.53162	0.531	0.5	0.672	930.3556	0.030501731	5.74%
50	0.53156	0.531	0.5	0.656	677.4464	0.026027801	4.90%
100	0.53562	0.531	0.515	0.672	888.8356	0.029813346	5.57%
200	0.54974	0.532	0.515	0.89	3528.5524	0.05940162	10.81%
500	0.54064	0.531	0.515	0.656	612.9104	0.024757027	4.58%
1000	0.5522	0.5465	0.515	0.922	3210.8	0.056663922	10.26%
1500	0.58092	0.547	0.516	0.922	6373.1936	0.079832284	13.74%
2000	0.60094	0.563	0.531	0.953	10790.2164	0.103875966	17.29%

Non-SDC Discovery

no. WS	arith. mean (in sec)	median (in sec)	min (in sec)	max (in sec)	variance (in msec^2)	stand.deviation (in sec)	coefficient of variation
10	0.5185	0.5705	0.094	1.094	80855.13	0.284350365	54.84%
20	1.1366	1.008	0.094	2.5	460362.4	0.678500111	59.70%
50	3.0282	2.867	0.219	5.844	2645127.48	1.626384788	53.71%
100	5.74968	5.7185	0.203	11.61	11150342.74	3.339212892	58.08%
200	12.9583	13.234	0.359	23.703	40064712.45	6.329669221	48.85%
500	31.36966	32.9465	2.922	60.095	292608442.9	17.10580144	54.53%
1000	56.52352	50.5715	3.563	163.44	1514250992	38.91337805	68.84%
1500	83.5551	77.142	5.875	182.612	2676860466	51.73838484	61.92%
2000	131.31316	150.894	3.719	248.975	5970983802	77.27214118	58.85%




Performance Comparison Goal Instance 6 (Median)
SDC Discovery
non SDC Discovery
Time (in sec.)
0
20
40
60
80
100
120
140
160
0
500
1000
1500
2000
2500
No. of Web Services


Performance Comparison Goal Instance 6 (Median)
SDC Discovery
non SDC Discovery
Time (in sec.)
0
1
2
3
4
5
6
7
0
20
40
60
80
100
120
No. of Web Services

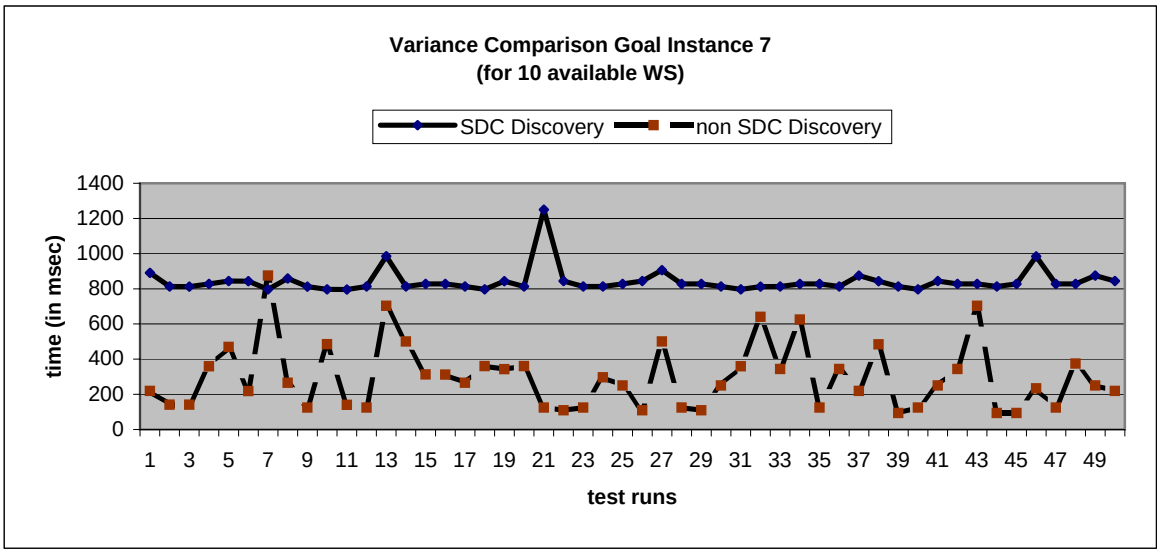
statistical preparation of comparison for: goal instance 7

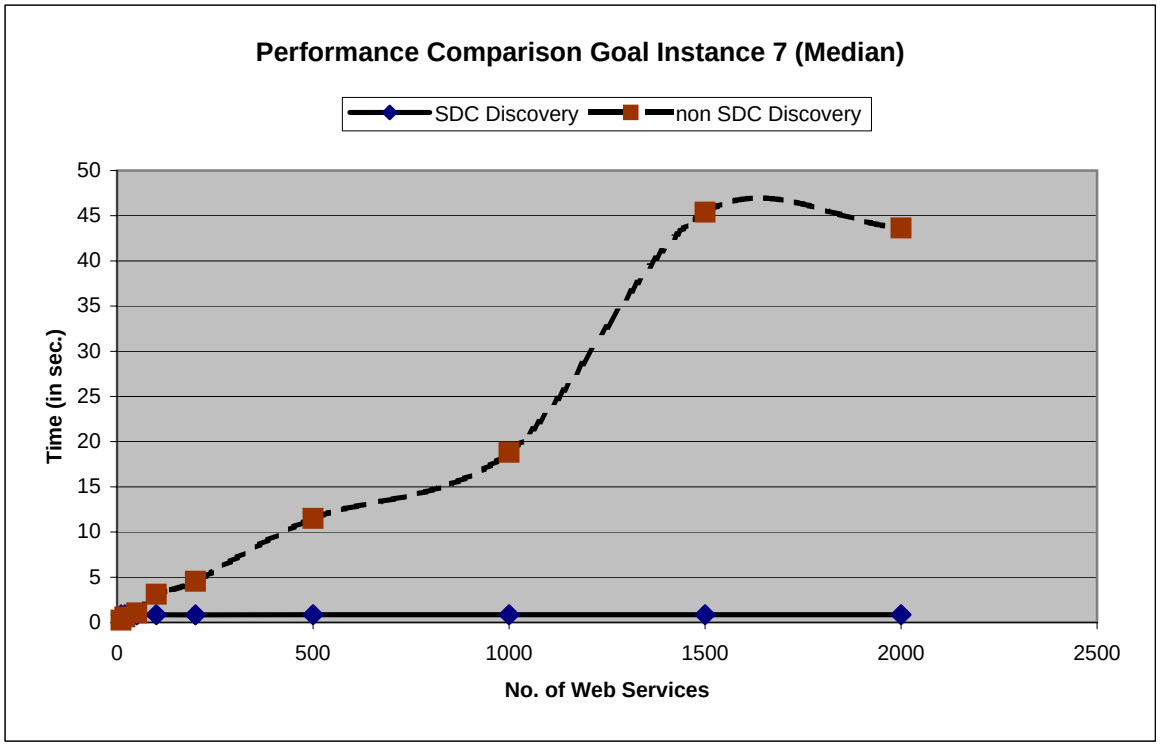
SDC Discovery

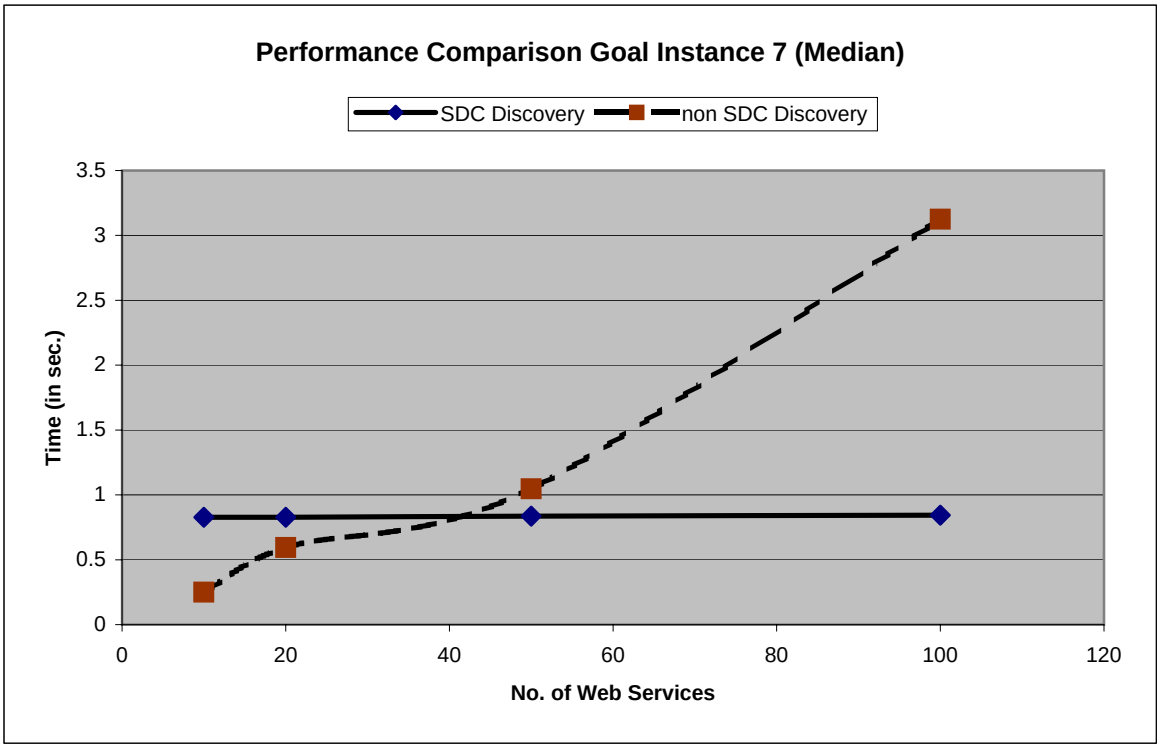
no. WS	arith. mean (in sec)	median (in sec)	min (in sec)	max (in sec)	variance (in msec^2)	stand. deviation (in sec)	coefficient of variation
10	0.84122449	0.828	0.796	1.25	4922.745523	0.07016228	8.34%
20	0.9419	0.828	0.796	5.735	474936.01	0.689156013	73.17%
50	0.85138	0.8365	0.797	1.265	4709.6756	0.068627076	8.06%
100	0.8665	0.844	0.812	1.25	6655.49	0.081581187	9.42%
200	0.86126	0.829	0.797	1.25	8959.1524	0.094652799	10.99%
500	0.8581	0.844	0.797	1.187	5358.37	0.073200888	8.53%
1000	0.8822	0.844	0.812	1.5	12153.68	0.11024373	12.50%
1500	0.86792	0.844	0.812	1.86	21356.3536	0.146138132	16.84%
2000	0.86184	0.844	0.812	1.172	3834.3344	0.061922003	7.18%

Non-SDC Discovery

no. WS	arith. mean (in sec)	median (in sec)	min (in sec)	max (in sec)	variance (in msec^2)	stand.deviation (in sec)	coefficient of variation
10	0.29532	0.25	0.094	0.875	33619.3776	0.183355877	62.09%
20	0.72626	0.594	0.094	5.625	636375.8724	0.797731705	109.84%
50	1.67348	1.0465	0.094	5.297	2124020.09	1.457401828	87.09%
100	3.46646	3.125	0.14	12.172	7054502.088	2.656031266	76.62%
200	5.6289	4.555	0.109	19.454	19586851.53	4.425703507	78.62%
500	16.04202	11.5005	0.234	54.939	168858004.1	12.99453747	81.00%
1000	26.03778	18.8285	1.359	83.877	426512815.8	20.65218671	79.32%
1500	48.69554	45.392	1.172	133.191	1122212051	33.49943359	68.79%
2000	58.73366	43.642	3.281	188.926	2382483636	48.81069182	83.11%




Performance Comparison Goal Instance 7 (Median)
SDC Discovery
non SDC Discovery
Time (in sec.)
0
5
10
15
20
25
30
35
40
45
50
0
500
1000
1500
2000
2500
No. of Web Services


Performance Comparison Goal Instance 7 (Median)
SDC Discovery
non SDC Discovery
Time (in sec.)
0
0.5
1
1.5
2
2.5
3
3.5
0
20
40
60
80
100
120
No. of Web Services

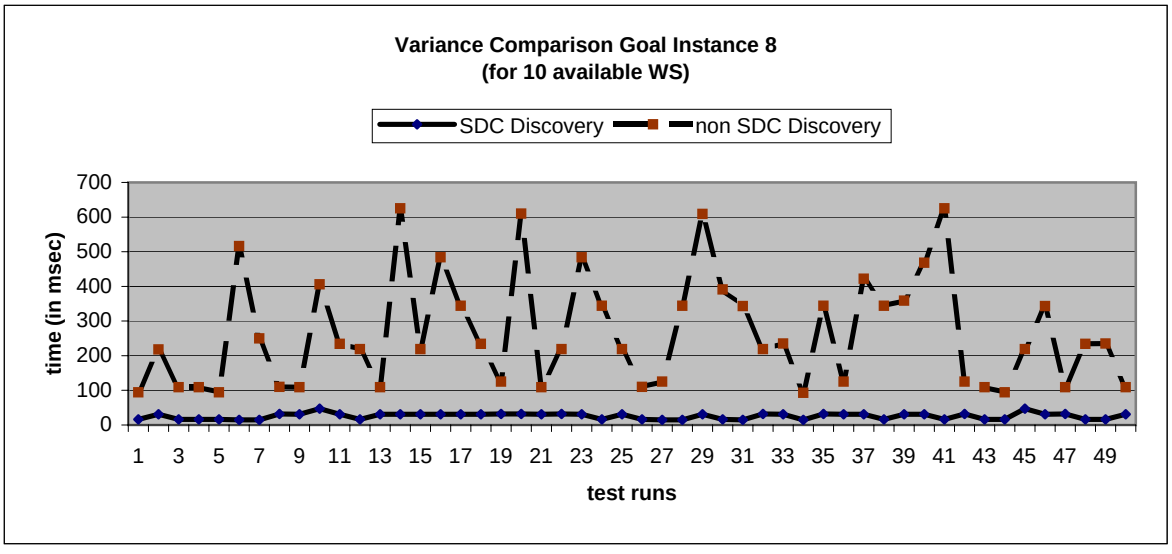
statistical preparation of comparison for: goal instance 8

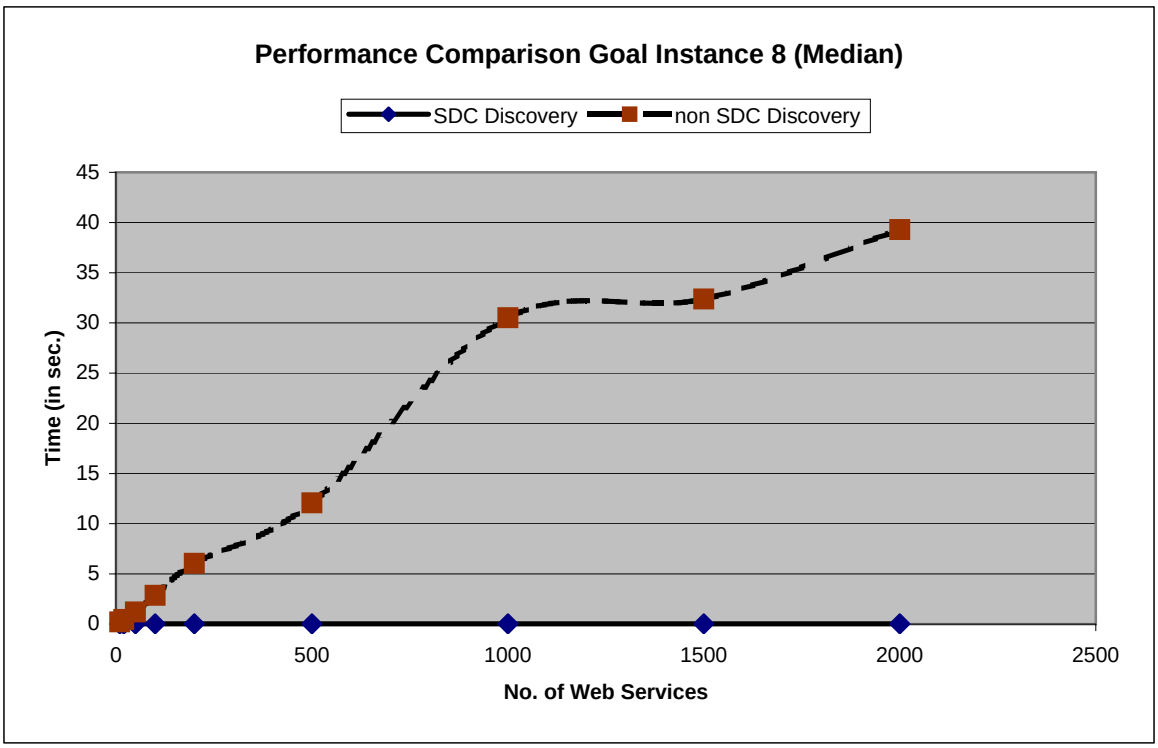
SDC Discovery

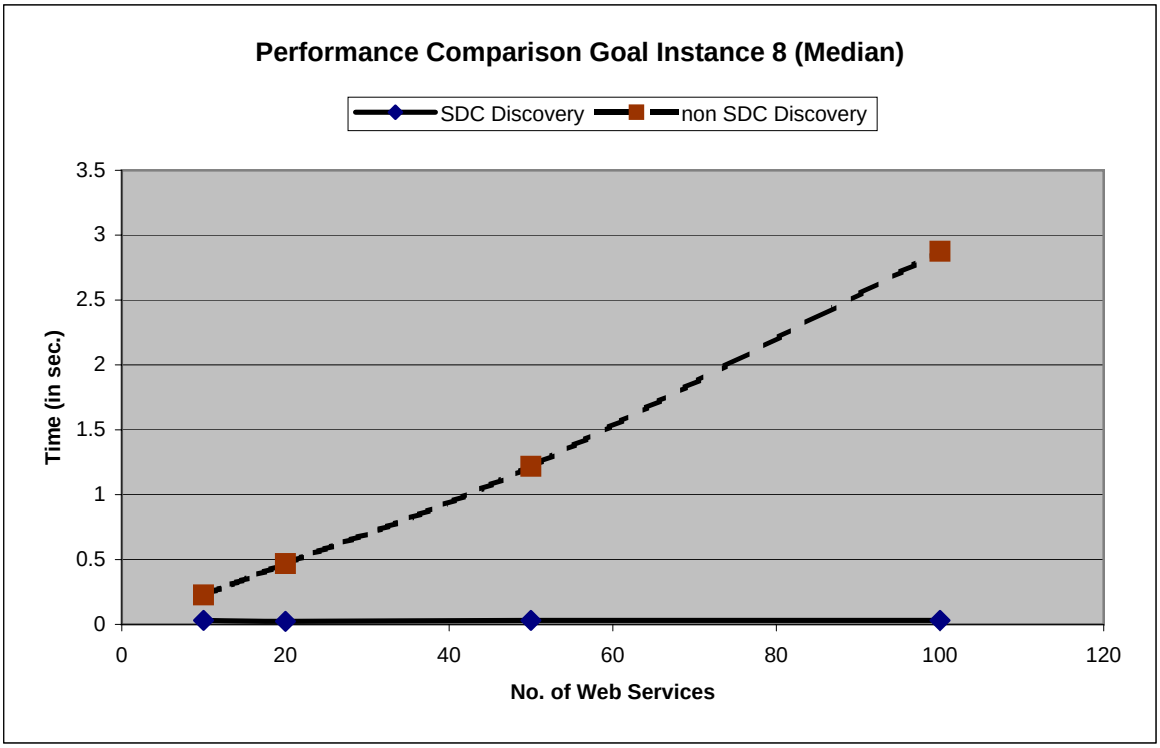
no. WS	arith. mean (in sec)	median (in sec)	min (in sec)	max (in sec)	variance (in msec^2)	stand. deviation (in sec)	coefficient of variation
10	0.02587755	0.031	0.015	0.047	75.41357768	0.008684099	33.56%
20	0.0269	0.0235	0.015	0.172	518.25	0.022765105	84.63%
50	0.03184	0.031	0.015	0.188	971.4944	0.031168805	97.89%
100	0.03034	0.031	0.015	0.172	888.1844	0.029802423	98.23%
200	0.02744	0.031	0.015	0.187	580.0064	0.024083322	87.77%
500	0.02438	0.031	0.015	0.032	61.5556	0.007845738	32.18%
1000	0.04302	0.031	0.015	0.422	5552.0596	0.074512144	173.20%
1500	0.03002	0.031	0.015	0.188	576.2196	0.024004575	79.96%
2000	0.02784	0.031	0.015	0.062	69.7744	0.008353107	30.00%

Non-SDC Discovery

no. WS	arith. mean (in sec)	median (in sec)	min (in sec)	max (in sec)	variance (in msec^2)	stand.deviation (in sec)	coefficient of variation
10	0.26652	0.2265	0.093	0.625	25349.0096	0.159213723	59.74%
20	0.63098	0.469	0.093	1.954	191615.2596	0.437738803	69.37%
50	1.59292	1.219	0.093	4.829	1377464.434	1.173654308	73.68%
100	3.20008	2.8755	0.109	10.485	4498163.754	2.120887492	66.28%
200	6.945	6.0625	0.25	16.001	19457440.68	4.411058907	63.51%
500	15.64352	12.071	0.125	50.892	166101445.2	12.88803496	82.39%
1000	35.34498	30.5165	1.578	107.69	664823392.7	25.78416942	72.95%
1500	40.12298	32.384	0.125	129.847	795477792.9	28.20421587	70.29%
2000	52.08724	39.298	0.485	187.817	2136735218	46.22483335	88.75%




Performance Comparison Goal Instance 8 (Median)
SDC Discovery
non SDC Discovery
Time (in sec.)
45
40
35
30
25
20
15
10
5
0
0
500
1000
1500
2000
2500
No. of Web Services


Performance Comparison Goal Instance 8 (Median)
SDC Discovery
non SDC Discovery
Time (in sec.)
3.5
3
2.5
2
1.5
1
0.5
0
0
20
40
60
80
100
120
No. of Web Services

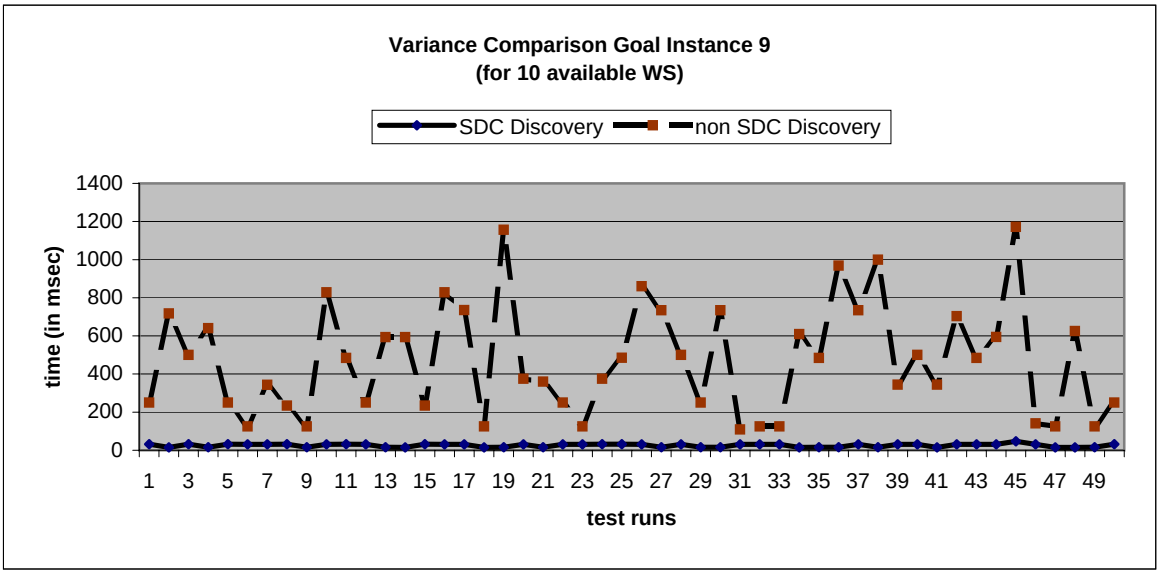
statistical preparation of comparison for: goal instance 9

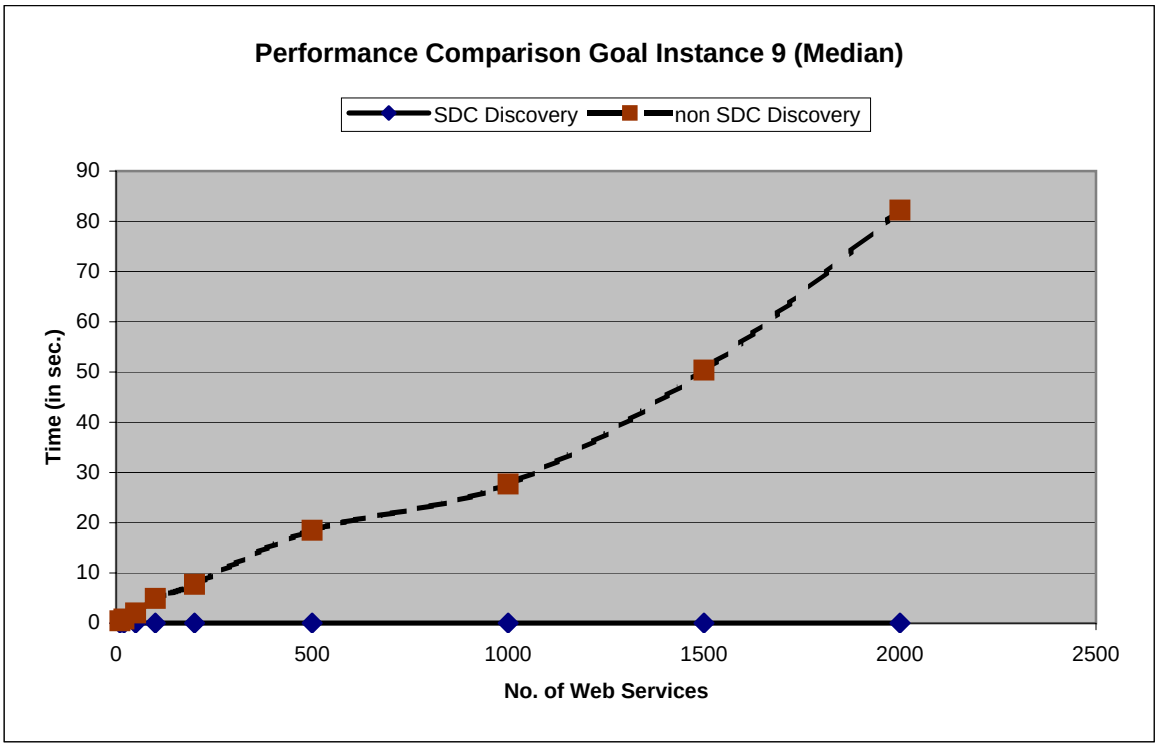
SDC Discovery

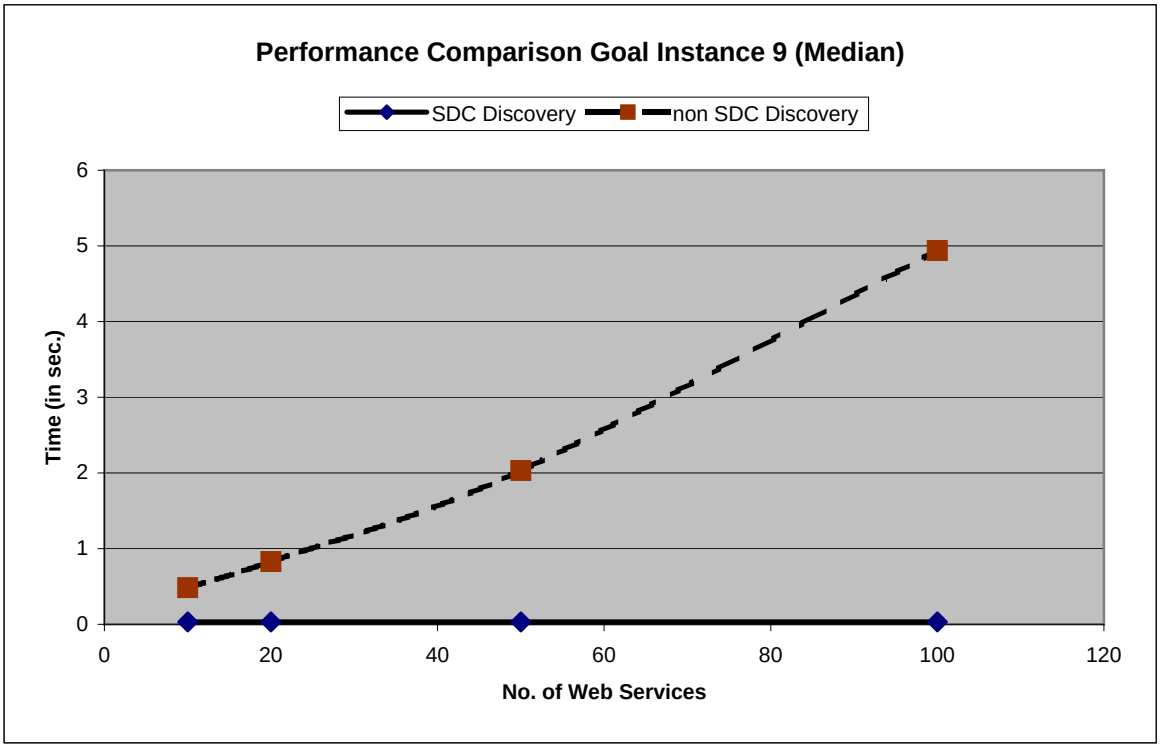
no. WS	arith. mean (in sec)	median (in sec)	min (in sec)	max (in sec)	variance (in msec^2)	stand. deviation (in sec)	coefficient of variation
10	0.02553061	0.031	0.015	0.047	67.14702207	0.008194329	32.10%
20	0.0321	0.031	0.015	0.172	914.69	0.030243842	94.22%
50	0.0275	0.031	0.015	0.047	54.65	0.007392564	26.88%
100	0.03152	0.031	0.015	0.157	711.3296	0.026670763	84.62%
200	0.02818	0.031	0.015	0.047	58.9876	0.007680339	27.25%
500	0.03058	0.031	0.015	0.14	299.8036	0.017314838	56.62%
1000	0.0385	0.031	0.015	0.406	3220.81	0.056752181	147.41%
1500	0.02872	0.031	0.015	0.062	110.3616	0.010505313	36.58%
2000	0.05164	0.031	0.015	0.954	17670.9104	0.132931977	257.42%

Non-SDC Discovery

no. WS	arith. mean (in sec)	median (in sec)	min (in sec)	max (in sec)	variance (in msec^2)	stand.deviation (in sec)	coefficient of variation
10	0.47244	0.484	0.109	1.172	81797.7664	0.286003088	60.54%
20	0.95972	0.828	0.125	2.25	381264.5616	0.617466243	64.34%
50	2.19534	2.0315	0.109	6.047	2036616.264	1.42710065	65.01%
100	5.18928	4.938	0.235	12	11579663.4	3.402890448	65.58%
200	8.79054	7.7345	0.109	23.532	32325589.53	5.685559737	64.68%
500	21.79904	18.4925	0.625	58.861	259218407.9	16.10026111	73.86%
1000	35.66096	27.721	0.468	108.314	951496707.5	30.84634026	86.50%
1500	60.73036	50.386	0.594	168.088	2425576340	49.25014051	81.10%
2000	90.47256	82.2385	0.125	229.966	3473247853	58.93426722	65.14%




Performance Comparison Goal Instance 9 (Median)
SDC Discovery
non SDC Discovery
Time (in sec.)
0
10
20
30
40
50
60
70
80
90
0
500
1000
1500
2000
2500
No. of Web Services


Performance Comparison Goal Instance 9 (Median)
SDC Discovery
non SDC Discovery
Time (in sec.)
0
1
2
3
4
5
6
0
20
40
60
80
100
120
No. of Web Services

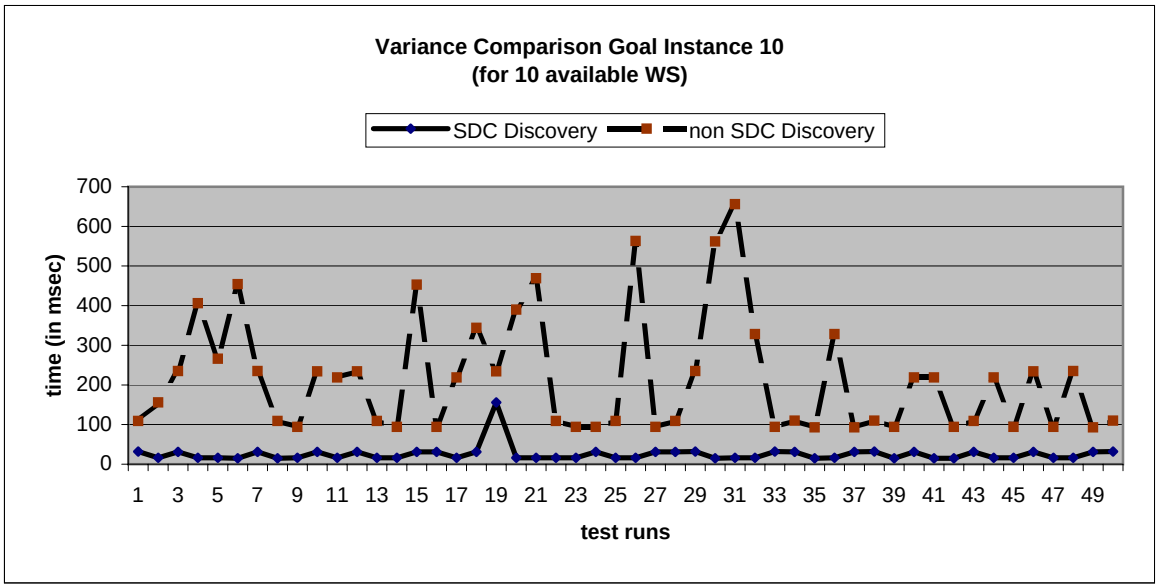
statistical preparation of comparison for: goal instance 10

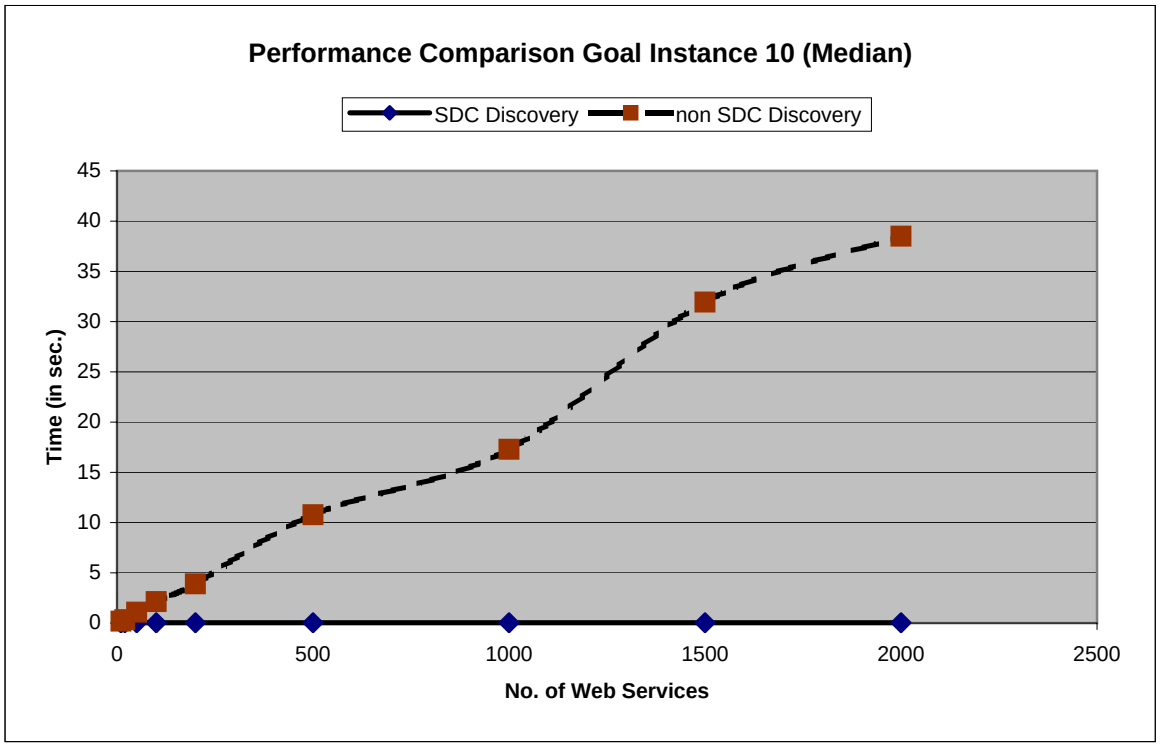
SDC Discovery

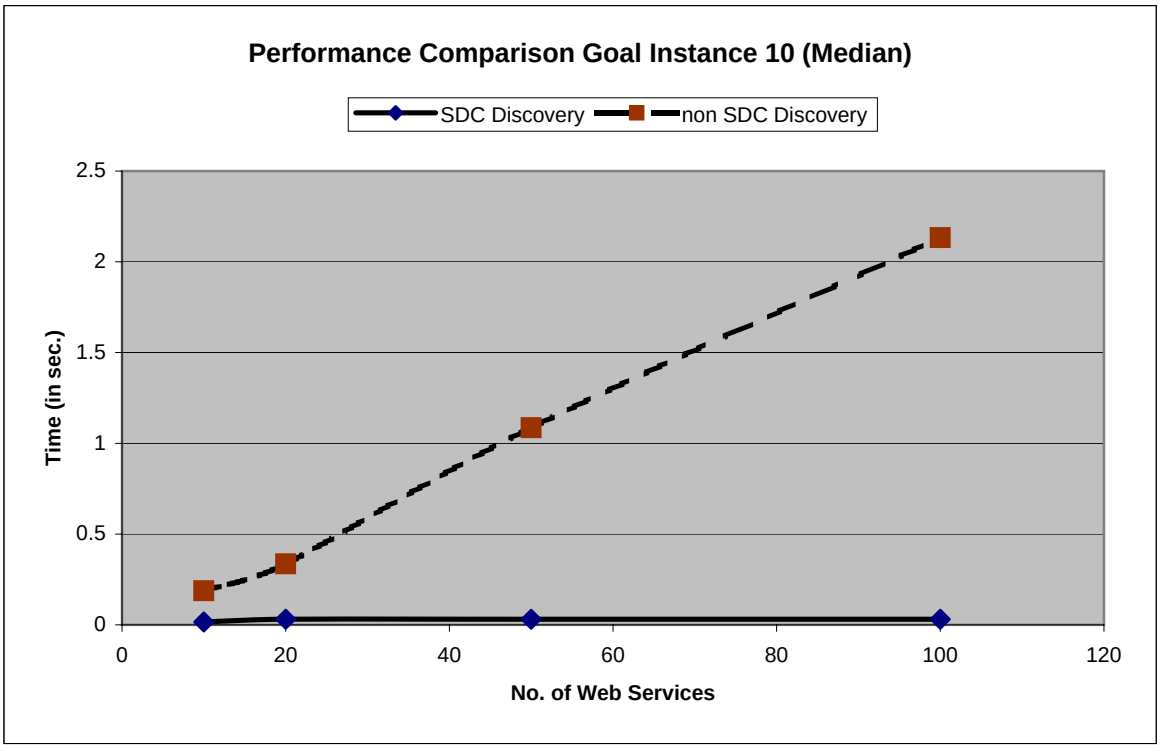
no. WS	arith. mean (in sec)	median (in sec)	min (in sec)	max (in sec)	variance (in msec^2)	stand. deviation (in sec)	coefficient of variation
10	0.02491837	0.016	0.015	0.156	414.9729279	0.020370884	81.75%
20	0.0278	0.031	0.015	0.157	397.88	0.01994693	71.75%
50	0.02432	0.031	0.015	0.032	60.0576	0.007749684	31.87%
100	0.03468	0.031	0.015	0.422	3111.6176	0.055781875	160.85%
200	0.03092	0.031	0.015	0.172	878.9536	0.029647152	95.88%
500	0.0513	0.031	0.015	0.609	10198.37	0.100986979	196.86%
1000	0.02906	0.031	0.015	0.188	568.9364	0.023852388	82.08%
1500	0.0322	0.031	0.015	0.203	1065.04	0.032634951	101.35%
2000	0.03214	0.031	0.015	0.187	822.4404	0.028678222	89.23%

Non-SDC Discovery

no. WS	arith. mean (in sec)	median (in sec)	min (in sec)	max (in sec)	variance (in msec^2)	stand.deviation (in sec)	coefficient of variation
10	0.21504	0.1875	0.093	0.656	21003.9184	0.144927287	67.40%
20	0.40372	0.3355	0.093	1.391	82090.5616	0.286514505	70.97%
50	1.25356	1.086	0.109	4.532	825091.3264	0.908345378	72.46%
100	2.24352	2.133	0.094	8.703	2374782.89	1.541033059	68.69%
200	4.62438	3.8985	0.094	12.282	13436127.68	3.665532386	79.27%
500	13.09228	10.782	0.5	36.658	96212509.92	9.808797578	74.92%
1000	23.94354	17.2895	0.734	86.723	450853532.6	21.23331186	88.68%
1500	34.15452	31.9395	1.094	128.445	821473537.8	28.66135966	83.92%
2000	48.4988	38.5105	1.625	123.101	1054707261	32.47625688	66.96%




Performance Comparison Goal Instance 10 (Median)
SDC Discovery
non SDC Discovery
Time (in sec.)
45
40
35
30
25
20
15
10
5
0
0
500
1000
1500
2000
2500
No. of Web Services


Performance Comparison Goal Instance 10 (Median)
SDC Discovery
non SDC Discovery
Time (in sec.)
2.5
2
1.5
1
0.5
0
0
20
40
60
80
100
120
No. of Web Services