Semantic Web Fred

SWF Architecture, Tools and Mechanisms

Net Dynamics & University of Innsbruck

Michael Stollberg
Uwe Keller
Ioan Toma
Peter Zugmann
Bernhard Keimel

October / November 2004

DERI – Digital Enterprise Research Institute

SWF-Deliverable:
D4

Version:
0.1

Date:
8-Nov-04

Executive Summary

This document specifies the technical realization of the SWF system with special attention to the SWF Discovery Mechanisms for establishing cooperative goal resolution. We specify the overall system architecture with special focus of the integration of the FRED technology and WSMO-enabled modules, and we specify the functionality, proof obligations, and architecture of the SWF Discovery modules for GG Discovery, GS Discovery, and WW Discovery as specified in the SWF Framework.

The Semantic Web Fred project is funded by the “Wiener Wirtschaftsföderungsfonds”, short: WWFF. The project partners are:

- Quarto Software GmbH, project leader
- DERI Austria at Institute for Computer Science at the University of Innsbruck, scientific cooperation partner.

Table of Contents

EXECUTIVE SUMMARY I
1 INTRODUCTION 6
2 SWF ARCHITECTURE 8
2.1 SWF Stepwise Development 8
2.1.1 SWF I and II: Prototype *9*
2.1.2 SWF III: Final *11*
2.2 SWF II Architecture (Prototype) 12
2.2.1 SWF Architecture Components *14*
2.2.1.1 System Setup 14
2.2.1.2 Task Delegation via Goal Instance Creation 15
2.2.1.3 Cooperation Establishment 15
2.2.1.4 Cooperation Execution in a FRED Meeting 16
2.2.2 Goal Management *17*
2.2.2.1 Structure of FRED Goal notions 17
2.2.2.2 Goals in SWF II 18
2.2.3 Ontology Management *18*
2.2.4 Resource Repository and Deployment *18*
2.2.5 Goal Solver and Discovery Management *18*
2.3 SWF III Architecture (Final) 19
3 IMPLEMENTATION OF WSMO DISCOVERY 20
3.1 General Discoverer Overview 20
3.2 Techniques applied for Discovery 23
3.2.1 Action Knowledge Discovery *23*
3.2.2 Object Knowledge Discovery *25*
3.2.2.1 Matchmaking Notions 25
3.2.2.1.1 Exact Match 25
3.2.2.1.2 Plugin Match 26
3.2.2.1.3 Subsumption Match 29
3.2.2.1.4 Intersection Match 30
3.2.2.2 Realization with VAMPIRE 32
3.2.2.2.1 Knowledge Representation 32
3.2.2.2.2 Proof Obligation 35
3.2.2.2.3 Strengths and Weaknesses of VAMPIRE 35
3.3 Modeling of Resources 36
3.3.1 Action Knowledge Description *36*
3.3.2 Object Description in WSMO Resources *38*
3.4 Conclusion 38

4 SWF DISCOVERERS 40
4.1 Implementation Overview 40
4.1.1 Discovery Manager 40
4.1.1.1 GG Cooperation 42
4.1.1.2 GS Cooperation 43
4.1.1.3 WW Cooperation 43
4.1.2 SWF Discoverer Modules 44
4.1.2.1 Discoverer Interfaces 44
4.1.2.2 Package Structure 44
4.1.2.3 VAMPIRE as Web Service 45
4.2 GG Discoverer 47
4.2.1 Overview 47
4.2.1.1 Discovery Request 47
4.2.1.2 Discovery Result 47
4.2.2 Functionality 47
4.2.3 Discovery Mechanisms Specification 48
4.2.3.1 Cooperation Knowledge Filter 48
4.2.3.2 GG Matcher 50
4.2.4 Interfaces 53
4.2.4.1 Input and Ouput 53
4.2.4.2 Resources 53
4.3 GS Discoverer 55
4.3.1 Overview 55
4.3.1.1 Discovery Request 55
4.3.1.2 Discovery Result 55
4.3.2 Functionality 56
4.3.3 Discovery Mechanisms Specification 57
4.3.3.1 Rationale of GS Discoverer Architecture 57
4.3.3.2 Pre-Selector 60
4.3.3.2.1 Service Class & Usage Permission Filter 60
4.3.3.2.2 Pre-Selector Matchmaker 61
4.3.3.3 GIS Matcher 61
4.3.4 Interfaces 62
4.3.4.1 Input and Output 62
4.3.4.2 Resources 63
4.3.4.3 Needed “Listeners” 63
4.4 WW Discoverer 64
4.4.1 Overview 64
4.4.1.1 Discovery Request 64
4.4.1.2 Discovery Result 64
4.4.2 Functionality 65
4.4.3 Specification 65

4.4.4 Interfaces *65*
4.4.4.1 Input and Output 65
4.4.4.2 Resources 66
5 CONCLUSIONS 67
REFERENCES 68
APPENDIX – SWF USE CASE OVERVIEW 69
A.1 Use Case Resources Overview 69
A.2 GG Discovery Matching Overview 72
A.3 GS Discovery Matching Overview 73
A.4 WW Discovery Matching Overview 74

Index of Figures

Figure 1: FRED- and WSMO Part of SWF 8
Figure 2: SWF I 9
Figure 3: SWF II 10
Figure 4: SWF III (final) 11
Figure 5: SWF Technical Architecture (SWF II) 13
Figure 6: FRED Goal Notions 17
Figure 7: Goal Management in SWF II 18
Figure 8: Discoverer - General Overview 21
Figure 9: Exact Match Overview 26
Figure 10: Plugin Match Overview 27
Figure 11: Subsumption Match Overview 30
Figure 12: Intersection Match Overview 31
Figure 13: SWF Discovery Manager Overview 41
Figure 14: Intermediate Results for Cooperation Establishment 42
Figure 15: GG Discoverer Overview 48
Figure 16: GS Discovery Overview 56

Index of Tables

Table 1: Matchmaking behavior PlugIn Match on Object Structure 28
Table 2: SWF Discovery Java Interfaces 44
Table 3: SWF Java Package and CVS structure 45
Table 4: Goal Templates Overview 69
Table 5: Goal Instances Overview 70
Table 6: Services Overview 71
Table 7: GG Discovery on Goal Templates (Cooperative Knowledge) 72
Table 8: GG Matcher – Matching Goal Instances 72

Table 9: GS Pre-Selector (Goal Templates <-> Services) 73
Table 10: GIS Matcher (Goal Instances <-> Pre-Selector Result) 73
Table 11: WW Discoverer Matchmaking (service compatibility) 74

Index of Listings

Listing 1: Universe Definition of a Generic Instance 33
Listing 2: Example Action Ontology 37
Listing 3: Example logical expression 38
Listing 4: WSDL Interface of VAMPIRE Web Service 46
Listing 5: GG Matcher Example 52
Listing 6: GS Discovery Matchmaking Behavior with Implication Match 58

1 Introduction

This deliverable specifies the SWF architecture for technical realization with special attention to the SWF mechanisms: GG Discovery, GS Discovery, and WW Discovery.

The preceding Deliverables of the SWF project have specified the conceptual model for (SWF D1: SWF Framework), and the techniques for semantic description of SWF components with special regard to the alignment with WSMO (SWF D3: SWF Goal and Service Description Language, v. 1). There have been several conceptual changes and adjustments in the past weeks, so that those Deliverables are not up to date anymore. However, an agreement on open conceptual issues has been found within the project consortium, which can be summarized in the following bullet points:

- The discovery mechanisms shall be completely and only WSMO-enabled, compliant to WSMO Standard, WSMO D2 final version 1.0 [Roman et al., 2004]
- The FRED part of SWF remains unchanged in the prototypical realization; the WSMO-part of SWF is linked with the corresponding FRED components via agreed use of specific links and relations; in the final architecture, the FRED-part will be adopted, resulting in a coherent, integrated, and WSMO-compliant architecture of SWF.

The SWF architecture describes the main components of the overall SWF system, their technical realization and interplay. We specify the final architecture, and outline how this is realized in first prototypical realization. The SWF Mechanisms are developed as open source modules (completely open at least for those parts which are WSMO-compliant) that are integrated into the overall SWF system structure. The resources along with a complete use case realization will be available on the SWF project website (http://www.deri.at/research/projects/swf/). The conceptual models for matchmaking in the distinct Discoverers are aligned with approaches developed within WSMO; the discovery in SWF works on WSMO only, thus the discovery mechanisms developed are fully compliant to WSMO Standard v1.0 as specified in [Roman et al., 2004]. We explain the objectives of the distinct discoverers, specify the matchmaking with the core proof obligations as well as the technological realization of each discovery along with the integration into the SWF overall architecture. For explanations within this deliverable, we refer to the SWF Use Case (a virtual marketplace for purchasing furniture), which is available at: http://www.deri.at/research/projects/swf/usecase/20040920/ .

The SWF Mechanisms *increase the rate of productive meetings*: a meeting between Freds is considered to be productive if at least the Goal of one participating Fred is resolved in course of a meeting – otherwise it is an unproductive meeting. The FRED system with the advanced discovery SWF Mechanisms is expected to be significantly higher than without them. This is the hypothesis to be falsified in an empirical testing after finalization of development, to be presented in the final project report.

The structure of this Deliverable is as follows: Section 2 specifies the SWF technical architecture with special intention on the integration and interplay of the FRED- and the WSMO-part of SWF; Section 3 explains the extended framework and the approach for realization of WSMO Discovery. Section 4 explains the conceptual model and the realization of the distinct discovery mechanisms, including a general overview of the approach and realization of WSMO Web Service Discovery realized in SWF, the architecture as well as specifications of the distinct SWF Discoverers; Section 5 concludes the paper.

This document is the first version of the SWF Deliverable D4 "Architecture, Tools and Mechanisms". It covers all aspects apart from WW Discovery; WW Discovery will be specified in a second version of this deliverable – as well as issues related to WW Discovery will be addressed in the second versions of other SWF Deliverables.

2 SWF Architecture

This section specifies the SWF Architecture. As many components of SWF are either existing in the FRED system or within specifications of the WSMO-part of SWF, we do not provide a specification of the complete SWF system, but focus on those aspects which are relevant for integrating the FRED-part and the WSMO-part of SWF.

Figure 1 shows the general separation of the WSMO-part and the FRED-part of SWF: While the latter is concerned with the FRED-specific issues like Goal Management, Goal Resolution Process, Service Execution and Meeting Management by using the technologies existing in the FredBase, the WSMO-part covers the discovery functionalities and the repository, providing the semantically enhanced mechanisms for establishing agent cooperation for cooperative goal resolution by WSMO-enabled technologies.

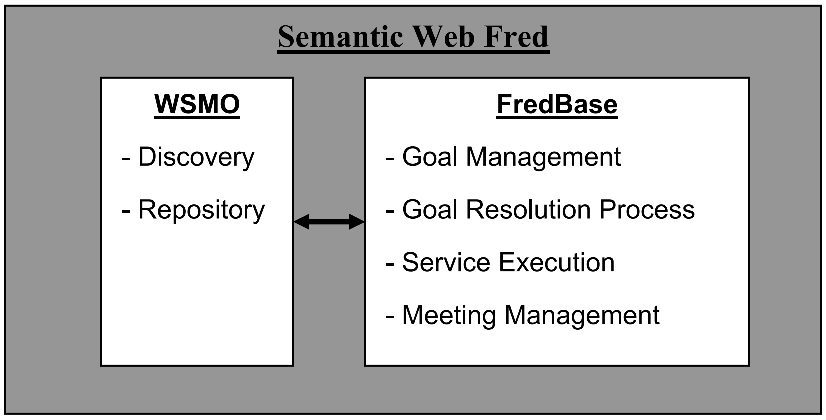


Figure 1: FRED- and WSMO Part of SWF

The overall aim of the SWF project is to develop a coherent system that integrates the FRED- and the WSMO part. This is done in a stepwise development, as explained in section 2.1.

2.1 SWF Stepwise Development

The realization of SWF consists of development of the SWF mechanisms in the one hand, and, one the other, of integrating WSMO-specific components with FRED-specific components. After the conceptual boundaries and differences have been discussed and resolved in preceding SWF Deliverables, this section concentrates on the technological part for realization.

WSMO is heavily developing, and the FRED-system shall be kept 'as unchanged as possible'. With regard to this, we define a stepwise development process for realization of SWF: so-called SWF I and SWF II define prototypes with different realization approaches, while SWF III provides the final, integrated architecture for SWF. The following outlines the approaches each SWF I, II, and III.

2.1.1 SWF I and II: Prototype

The first scenario of a SWF prototype realization is a parallel usage of a WSMO system for those aspects of SWF that shall be realized by WSMO, for examples WSMX [WSMX], and the FRED system for those aspects that reside FRES-specific within SWF. While the FRED-part (right-hand side) controls the goal resolution process, an external WSMO system is used for discovery and related aspects (mediation) for establishing agent cooperation. Inside the FRED-part the existing FRED technologies are applied, there are no changes; and the WSMO-part works with WSMO elements only (goals, services, mediators, and ontologies). For interchange between the systems, complex transformations as well as control structures need to be defined (deploying FRED Goals and Services as WSMO services, interchange of discovery results and an overall system control structure). This approach is referred to as SWF I, Figure 2 shows the general setup of this architecture.

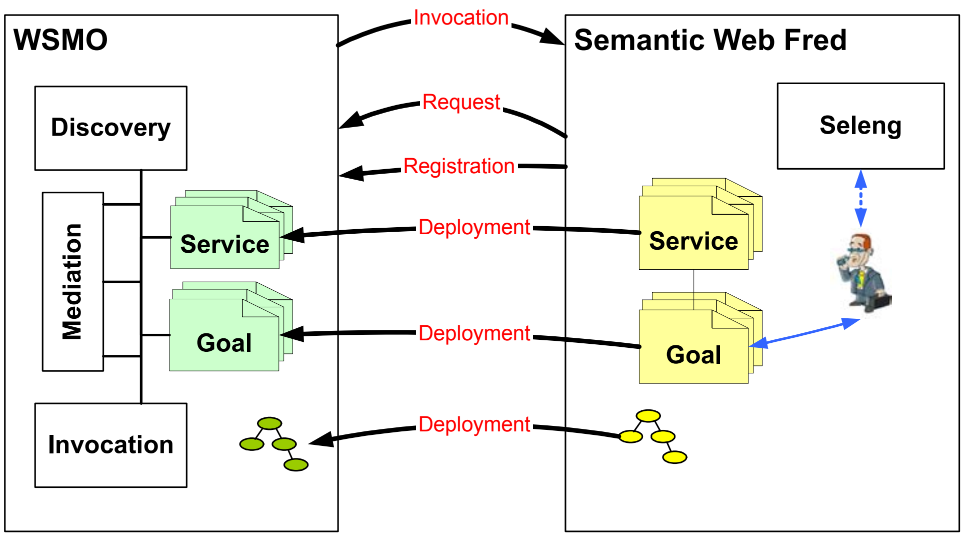


Figure 2: SWF I

Although this architecture allows making us of WSMO technologies, a major drawback is that it is not controllable by SWF as core components are "outsourced". Besides, by the time of development, there is no WSMO-enabled system existing that can provide the required facilities in a stable and secure manner (WSMX is still under development).

Thus, a different approach is taken for realization of the prototype, called SWF II: the required WSMO-enabled facilities for discovery are developed as modules that can be incorporated into the SWF system, so that the complete system can be controlled and maintained by SWF.

As within SWF I, the FRED-part of SWF controls the cooperative goal resolution process with the existing FRED technologies, i.e. without changing the FredBase. The WSMO-part is working on WSMO-compliant components of SWF only, applying SWS technologies developed within WSMO. The major benefits of SWF II in comparison to SWF I am that the complete system functionality can be controlled within SWF, second that the discovery mechanisms can be designed precisely to the needs of SWF, and third that functional incompatibilities between the SWF-parts can be avoided. Figure 3 gives an overview of this architecture.

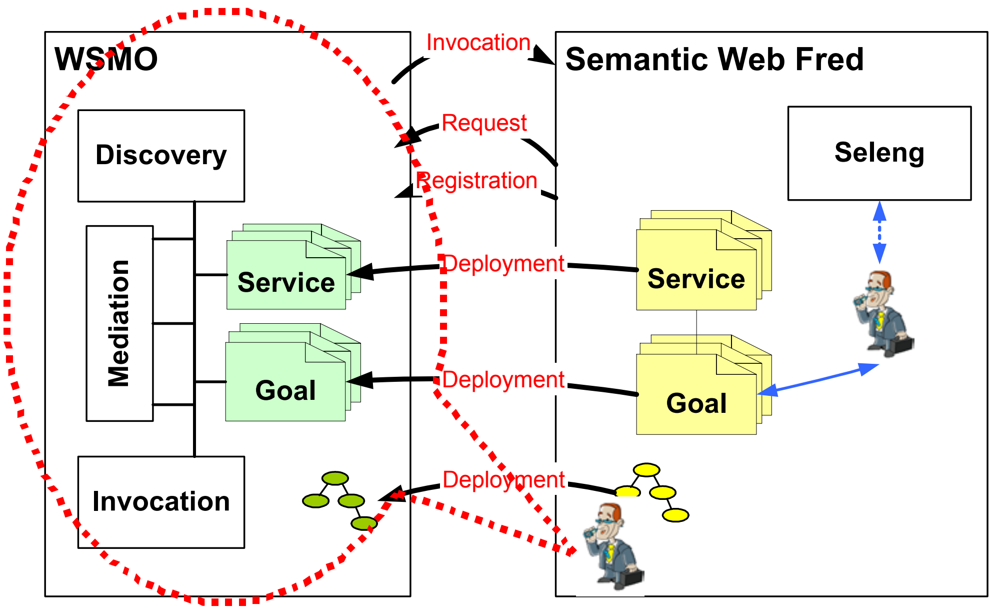


Figure 3: SWF II

Although this architecture overcomes the problems of dependency on an external, not controllable system for core functionalities, SWF II still requires interaction between the FRED- and the WSMO-part: transformations of ontologies, as well as management of goals and services within and in between the SWF parts.

SWF II has been chosen for developing the SWF prototype because of the issues discussed. A main aspect of the prototype architecture was to not require fundamental changes in the FRED system: as FRED is a complex system and is in use for different applications, the changes within the FRED system are postponed to the final architecture when the WSMO-part is developed in a stable version.

2.1.2 SWF III: Final

The step from the SWF II prototype to the final architecture is to change the FRED-part of SWF according to the functional requirements of the WSMO-part, so that the final SWF architecture represents an integrated system of the FRED and WSMO functionalities combined within SWF.

In order to make the FRED-system compatible to the WSMO-part of SWF, the FRED following FRED components have to be re-designed: ontologies need to be exchangeable with WSMO-system (whatever format / representation is chosen), and FRED Goal and Service description have to be appended with those WSMO notions required by the WSMO-part of SWF (i.e. the discoverers). In doing so, the transformations and the doubling of resources as needed within SWF II are redundant, and the final SWF architecture is WSMO compliant, thus resources can be interchanged with other WSMO-enabled system. Figure 4 shows this architecture.

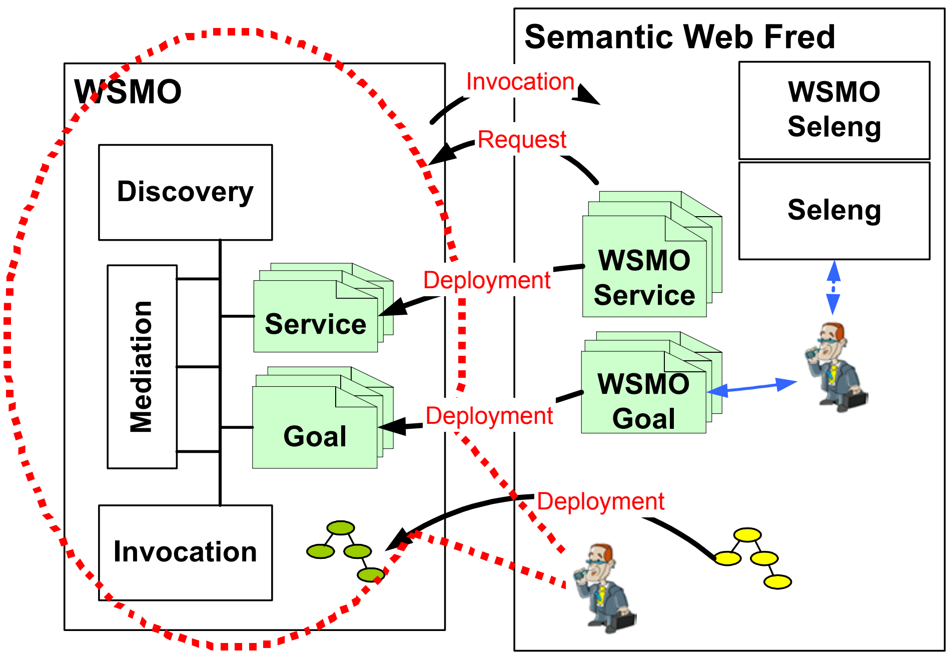


Figure 4: SWF III (final)

The required changes in the FRED will result in a new version of the FRED system. Obviously, these changes will have substantial implications on existing FRED technologies which can not be precisely and exactly identified at this point in time. The concrete requirements and realization for the required changes will be specified at a later stage of the project.

Summarizing, the final SWF architecture (SWF III) provides:

- Integrated system for cooperative, agent-driven goal resolution for WSMO resources with dynamic service usage
- WSMO-enabled mechanisms for advanced cooperation establishment mechanisms
- Mature overall system control
- Interchangeability of Ontologies, Goals, and Services with other WSMO-enabled systems.
- A new, enhanced FRED system

2.2 SWF II Architecture (Prototype)

As outlined above, with SWF II the SWF Discoverers as the core components for cooperation establishment are developed and incorporated with the existing FRED technologies. Apart from the needed integration and the necessary changes within the FRED system, SWF II represents the final architecture of SWF.

The following explains those parts of the SWF II architecture that are related to the development of the WSMO-enabled SWF Discoverers and their integration within the existing FRED technologies. Figure 5 provides an overview of the SWF II architecture; below, we explain the components and their interrelation as well as specific aspects in more detail.

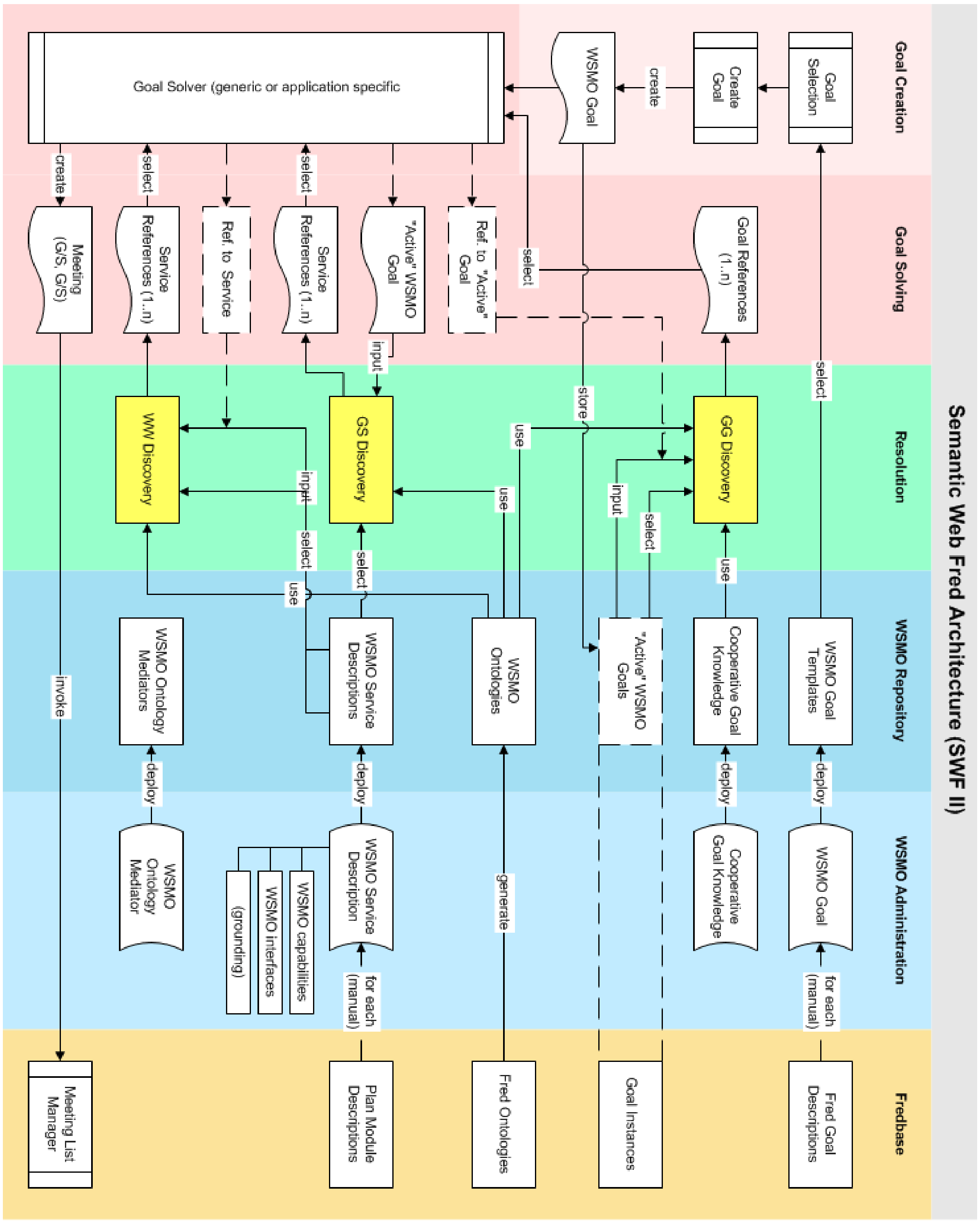


Figure 5: SWF Technical Architecture (SWF II)

2.2.1 SWF Architecture Components

Figure 5 allocates the SWF components to their functional area, denoted by colored backgrounds; starting from the right hand side:

- FredBase (yellow): the FRED-part of SWF, existing FRED components
- WSMO Administration (light blue): deployment of FRED components in to WSMO components
- WSMO Registry (blue): SWF resource repository, which is aligned with the WSMO Registry [Herzog et. al, 2004]
- Resolution (red): SWF Discoverers and Cooperation Establishment Control
- Goal Creation (light red): Goal Creation, i.e. assigning a task to a Fred-agent for automated resolution

We provide a walk-thru the system for explanation, starting at the upper right corner with "Fred Goal Description".

2.2.1.1 System Setup

The first phase in SWF is System Setup, wherein the resources needed are created. These are Ontologies, Goal Templates, and Services.

In the FRED-system, there exists the notion of 'Goal Descriptions' and 'Goal Instances'. Form the functional point of view, these notions are identical to Goal Templates (Goal Templates) and Goal Instances in SWF – apart from that they do not carry the WSMO notions needed for discovery. For system setup, FRED Goal Description and FRED Goal Instances as well as their corresponding WSMO elements have to be created. This is described in more detail in section 2.2.2.

Ontologies have to be created as the terminology specifications used in the other components; in SWF, Smart Objects are used within the FRED-part, and WSMO ontologies are required by the discoverers. The techniques for ontology management within SWF are explained in detail in section 2.2.3. A special type of ontology is the Cooperative Knowledge Ontology which specifies compatible Goal Templates with regard to the cooperation role and the object of interest of the Goal Templates; this ontological knowledge is defined at system setup and used within GG Discovery (see section 4.2.3.1).

SWF Services consist (as every Web Service) of the "real program" and the Service description. The "program"-part is defined within the FredBase – as a Plan, Process, or an external Web Service; the service description is done manually in, describing the service as a WSMO Web Service. The deployment of FRED resources into the WSMO-part of SWF is explained in section 2.2.4.

2.2.1.2 Task Delegation via Goal Instance Creation

After setup, the system functionality is initiated by creating Goal Instances and assigning them to Freds for automated resolution. Goal Instances are created by 'instantiating' a Goal Template, possibly refining the desire and defining the additional elements of Goal Instances. While in SWF II, Goal Instances have to be created manually, a graphical user interface will support Goal Instance creation in the final version of SWF.

The following rules hold for Goal Instance Creation:

- **Goal Instances can only be created out of Goal Templates existing in the system.** This ensures that there are no Goal Instances in the system that can not be resolved because no services for resolution are available.
- **The desire expressed in a Goal Instance can only be the same as in the corresponding Goal Template or a refinement of this, but not an extension.** A Goal Instance is understood as an instantiation of a Goal Template, a concrete desire; the design of the SWF Discoverers is based on this rule.

Within SWF II, Goal Instances are represented by FRED Goal Instances which have assigned an "Active WSMO Goal". An Active WSMO Goal is a WSMO Goal that specifies the desire of the Goal Instance within WSMO notions. The reason for not using SWF Goal Instances here but WSMO Goals (they have less descriptive information: a SWF Goal Instance defines additionally the submission as data that are intended to be submitted as input to a service) is to allow the SWF Discoverers to be completely WSMO-compliant. This might be changed in future versions of SWF.

2.2.1.3 Cooperation Establishment

Then a cooperation between Freds is established by different discovery mechanisms:

- **GG Discovery:** find potential cooperation partners by determining compatibility of Goal Instances
- **GS Discovery:** find usable services for each potential cooperation partner as determined by GG Discovery.

- **WW Discovery:** out of the result from GS Discovery, determine one Service to be used for automated cooperative Goal resolution for each partner determined in GG Discovery.

The SWF architecture overview in Figure 5 shows the location, input, resources, and outputs of the discoverers. The architecture and functionality of the distinct discoverers is specified in Section 4of this document.

The **SWF Goal Resolution Process** located in the **Goal Solver** comprises the control of the resolution of Goal Instances that the Freds participating in a cooperation are assigned, and the control of the SWF Discoverers. Section 2.2.5 described the Goal Solver in more detail, while the SWF Discovery Manager is specified in section 4.1.1.

2.2.1.4 Cooperation Execution in a FRED Meeting

The result of Cooperation Establishment is a Cooperation Contract that holds all information that is needed to execute the cooperation in a Meeting Room. Such a Cooperation Contract is comprised of:

- 1 .. n Freds
- Goal Instances of the Freds (each participating Fred has 1 Goal Instances)
- Services (n Services for each Goal Instance)
- Needed resources:
  - Ontologies used in Goal and Service Description
  - Mediators used in Goal and Service Description, and those invoked within some Discoverer (this is only conceptual, not realized in SWF)

The Goal Solver dynamically establishes Cooperation Contracts after successful cooperation establishment, and defines FRED Meetings for cooperation execution. These meetings are executed in Meeting Rooms as existing in the FredBase.

The most basic SWF service type is a Plan, a Java program which encompasses the communication and information interchange facilities as FIPA communicative acts; service execution is performed in a FRED Meeting Rooms by executing the Plan Java programs along with the communication and information interchange as defined within the Plan.

2.2.2 Goal Management

Within SWF II, a "duplication" of the notions for SWF Goals is needed: FRED Goal Description and Instances are needed within the FRED-part for goal resolution process management, and WSMO Goals are needed for the WSMO part for discovery.

Form a functional perspective, FRED Goal Description are similar to SWF Goal Templates (i.e. pre-defined desires which a Goal Instance can be created from), and FRED Goal Instances are equal to SWF Goal Instances (concrete desires assigned to a Fred-agent for automated resolution). For SWF II, these FRED Goal notions are used for goal resolution process management in the FRED-part of SWF, and each FRED Goal notion is coupled with a corresponding WSMO Goal in order to support the SWF Discovery mechanisms.

To make this more explicit, we first recall the structure of the FRED Goal notions, and then specify the relation and handling of "duplicated Goals" in SWF II.

2.2.2.1 Structure of FRED Goal notions

The structure of Goal notions in FRED looks like this:

(to be a nicer figure)

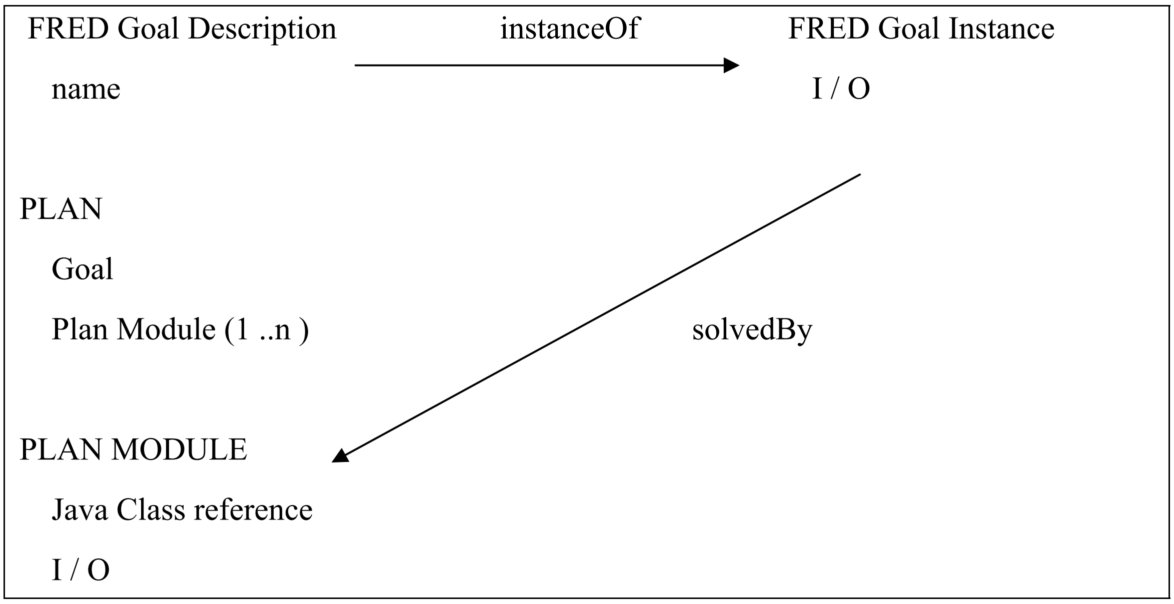


Figure 6: FRED Goal Notions

The idea / aim of SWF: replace the notion of a Plan by advanced discovery mechanisms.

2.2.2.2 Goals in SWF II

In SWF II, the FRED Goal notions are kept for the FRED-part of the system, and in addition they are linked to corresponding WSMO Goals to support the WSMO-enabled SWF discoverers. Figure 7 shows the definition of the relation and the connections between the SWF II goal notions.

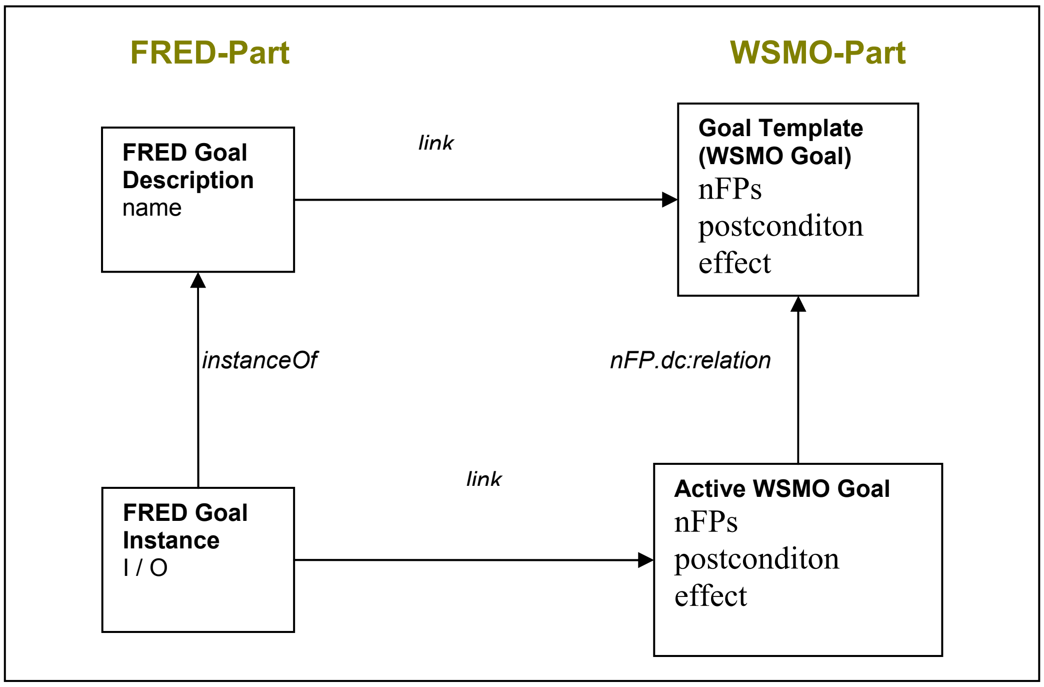


Figure 7: Goal Management in SWF II

Explanations …

2.2.3 Ontology Management

Transformation FRED Ontologies (<)-> WSMO Ontologies

To be added -- sufficient resources existing

2.2.4 Resource Repository and Deployment

SWF - WSMO Registry / Repository & resource deployment process

To be added -- sufficient resources existing

2.2.5 Goal Solver and Discovery Management

Description / Spec of the Goal Solver, which basically does 4 things:

1) Control / Monitoring of the resolution process / status for Goal Instances that are owned by Fred participating in a cooperation
2) Control & Invoke the SWF Discoverer Modules, collect the discovery results and dynamically create the overall discovery result. The latter is called the SWF Discovery Manager as specified in section 4.1.1.
3) Provide access and manage the resources needed by the discoverers
4) Create a Cooperation Contract in the end and define this as a FRED Meeting for Cooperation Execution

2.3 SWF III Architecture (Final)

Necessary changes from SWF II to SWF III:

Basically, the existing FRED notions need to be expanded by the additional notions of WSMO Goals and Services, and the ontology transformation technology has to be enhanced.

This has several impacts which can not be foreseen and estimated at this point of time; here is an initial list of aspects:

- maybe some existing FRED technologies / mechanisms are affected by these changes
- some additional components would be needed to enable specification and processing of WSMO / WSML

These requirements will be investigated after the SWF II prototype is finalized, as specific issues will more obvious then.

3 Implementation of WSMO Discovery

In accordance to the generality of WSMO as an open and overall framework for Semantic Web Services, Web Service Discovery within WSMO does not provide a concrete solution but rather a framework of discovery approaches. Different possibilities, ranging from key-word matching over controlled vocabulary matching and ontology-based matchmaking up to full fledged matchmaking on basis of logical expressions are identified as possible WSMO Web Service Discovery solutions. These are interrelated with the kind of resource descriptions of service usage requests and Web Services, whereby matchmaking on basis of logical expressions obviously promises the highest quality of discovery results [Keller et al., 2004].

Within SWF, Goals and Services are described as WSMO resources, thus they are described by clearly defined logical expressions, thus the realization of the matchmakers within distinct SWF Discoverers provides a prototypical solution for WSMO Web Service Discovery based on simple semantic descriptions of Services as defined in section 4.2 [Keller et al., 2004]. For the WSMO components and their description we refer to WSMO version 1.0 through the whole project [Roman et al., 2004]

The remainder of this section gives a general overview of the realization of discovery mechanism. We explain the general overview of a discoverer, the types of discovery techniques and the modeling of resources required for our approach, and an overview of the technological realization of discovery. We provide a detailed elaboration of a framework and realization for discovery, covering conceptual, theoretical, and realization-related aspects. Thus, the following can be seen as an extension to the general, theoretic framework provided within [Keller et al., 2004].

3.1 General Discoverer Overview

Starting very general, a discoverer looks like this: a Discovery Request is provided as input, then a specific architecture of filters and matchmakers determines the matching resources, and a Discovery Result is returned that contains the resource that match with the Discovery Request. Figure 8 shows this general structure.

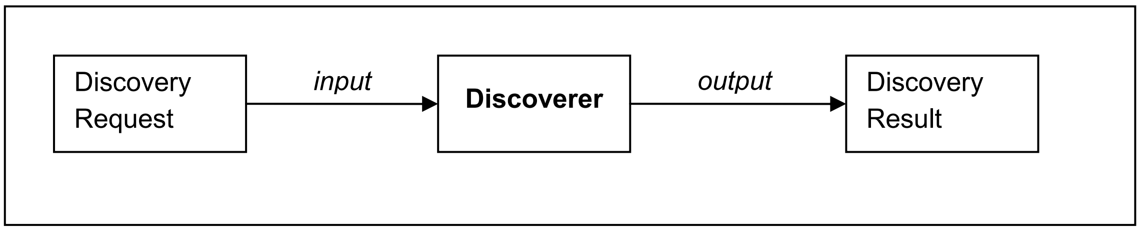


Figure 8: Discoverer - General Overview

To allow a discoverer to return high quality results with respect to the expected discovery functionality, we have to provide it with as much knowledge as possible. Therefore, we extend the discovery framework with more detailed definitions of the in- and output by the notions of Discovery Request (the input) and Discovery Result (the output).

The more generic terms of Discovery Request (the input) and Discovery Result (the output) extend the notions of Goal as input and Web Service as output defined in [Keller et al., 2004].

A Discovery Request contains all the information that is required for the Discoverer to provide his functionality, and the Discovery Result contains the matching resources detected as well as the definition of the relation to the Discovery Request. In principle, the Discovery Request and Result can be related to any WSMO top level component; for example, for the SWF GG Discoverer (see section 4.2) the Discovery Request as well as the Result are WSMO Goals, for the GS Discoverer the request is a WSMO Goal and the result is matching WSMO Services, and for WW Discovery, both the request and the result are WSMO Services. The management of Discovery Requests and Results in a system with several sequentially interacting Discoverers can be kept within a specific component like the SWF Discovery Manager (see section 4.1.1), thus modularizing the design of specific components. The following outlines the underlying approach for realization of Discovery for Web Service Discovery, i.e. detecting of potentially usable services for solving a Goal.

In a Discovery Request for Web Service Discovery, a Goal is provided as the basic input. For the realization of the discoverer, we differentiate specific aspects of the information specified in a WSMO Goal. The main distinction is between **Actions** and **Objects**: an Actions specifies what shall be performed with an Object, and the Object specifies the item to be handled. The relation between these notions is: Action(Object). This approach relies on previous work for realization of Web Service Discovery [Grimm et al., 2004].

An example for explanation, referring to the SWF Use Case which defines a virtual marketplace for buyers and sellers to purchase furniture [Stollberg, 2004]. Actions in this marketplace are ‘buy’, ‘sell’, ‘deliver’, ‘search for production information’, etc., and the Objects are the products / furniture that are to be sold / bought by marketplace participants. A Goal “buy 1 brown armchair” is understood as buy (brown armchair) with the action ‘buy’ and the object ‘brown armchair’, similar a Service “sell armchairs” is understood as sell (armchair).

The reason for this distinction is that it seems not appropriate to encode the information on actions and objects on the same level of ‘application logic’, as it is hard to determine which aspects of a logical expression describe an action and which ones the object. Not providing this distinction would have negative impacts: first, the modeling of resources with an application would be restricted in the way that a resource description corresponds to the way the discoverer ‘understands’ or handles resources, and secondly the matchmaker would have to provide complex facilities to differentiate between the action and object notions to prevent detection to not fitting resources in the Discovery Result. The discovery techniques for Actions and Objects are different, as explained below.

Besides the distinction of Actions and Objects, a Discovery Request has to provide the Matchmaking Notion that shall be applied by the Discoverer for detecting matching resources. The reason is there are different matchmaking notions whose applicability depends on the discovery intention, i.e. what precisely a request means and by which relationships resources are considered to be matching. The Matchmaking Notions are different for discovery on Actions and Objects, further there are different matchmaking notions for each of these as presented below.

Summarizing, within our extended framework a Discovery Request consists for three different aspects which together provide the knowledge needed for discovery as input to a discoverer. These notions are independent of the WSMO top level notion the request is aligned with.

1) **Action**

   An Action defines an activity that shall be performed or achieved on an Object, the relation of Action and Object is Action(Object).

   Actions are usually provided within the description of a WSMO component, but are not marked explicitly. By providing the Action within a Discovery Request, we mean that the knowledge about Actions has to be provided explicitly in addition to the basic WSMO top level notion that the request is aligned with.

2) **Object**

   The Object is the item that an Action is performed on.

   Similar to Actions, Objects are usually provided within the description of a WSMO component. In contrast, Objects do not have to be provided explicitly in addition to the WSMO component, as the required knowledge of the relationship of Actions and Objects in a Discovery Request is already provided within the explicit declaration of the Action items.

3) **Matchmaking Notion**

   The Matchmaking Notion to be used by the discoverer has to be explicitly provided within a Discovery Request in order to specify under which relationship resources are considered to be matching the Discovery Request. There are different types of Matchmaking Notions as discussed below.

A Discovery Result is the general term for resource types that are determined to be matching the Discovery Request. The Discovery Result is related to a WSMO top level notion, e.g. to Web Services for Web Service Discovery or to Goals within other types of discovery like the GG Discoverer in SWF (see section 4.2).

In order to allow a discoverer to process a Discovery Request as described above, the WSMO top level notion that the requested Discovery Request is aligned with needs to be described with notion corresponding to the notions distinguished within the Discovery Request. In particular, this means that the Action items of the WSMO top level notion description have to be provided explicitly, while the Objects are identified inherently by the component description. Regarding the support of the Matchmaking Notion, either the matchmaking notion supported/ intended for the resource need to be provided explicitly, or the semantics of the description notions need to be clearly defined.

3.2 Techniques applied for Discovery

For the types of knowledge items differentiated above, different techniques for discovery are applied with regard to the structure of the specific ontology item and techniques that seem to be most appropriate for processing these information.

3.2.1 Action Knowledge Discovery

The aim of discovery on Actions is to determine which parties should interact via their Goal, Web Services, or another WSMO top level component that they ‘own’. For example, when a buyer *B* has the Goal “buy an Object O” and a seller *S* provides a Web

Service for "sell instances of Object O", the discovery on the Action knowledge should determine that *B* and *S* should interact.

So, discovery on Action items is concerned with *determining the compatibility of actions*. In the example, the actions 'buy' and 'sell' are compatible in the domain of discourse. In order to detect this, there needs to be an appropriate description of the actions in the domain of discourse along with their compatibility, and there needs to be an appropriate mechanism that determines the compatibility of WSMO resources on basis of their action descriptions.

Actions can most likely be described in special ontologies which define the compatibility of actions, here referred to as *action knowledge ontologies* (see 3.3.1). On this, filters can be defined which determine the action compatibility of WSMO resources. A naïve, straight forward way for realization of such action knowledge discovery filters is mechanisms realized in conventional, i.e. not inference based technologies that work on the action knowledge ontology via APIs such as WSMO4J[1]; depending on the complexity of the action knowledge ontologies, inference based techniques might become necessary with regard to the expressiveness required to determine action compatibility. Obviously, the former solution usually is faster than the usage of inference technologies due to technical maturity of existing platforms.

As discussed within the realization of GG Discovery in 4.2.3.2, action knowledge discovery is needed in order to ensure correct discovery results. Without determining the action compatibility, WSMO resources might be in the Discovery Result which – although modeled 'correctly' with respect to what should be expressed – cannot satisfy the intention of the Discovery Request.

Regarding the internal organization of the discoverer, the most expensive operation in terms of performance should be performed at last, i.e. on the smallest set of resources that have to be matched.

Within the SWF Use Case [Stollberg, 2004], discovery on objects by inference-based matchmaking is more expensive than action knowledge discovery. Thus, action knowledge discovery is performed before object knowledge discovery, and can be seen as a pre-selection that minimizes the set of possibly relevant resources wherefore the matching has to be tested by object knowledge discovery. This might be regarded as a general guideline, as object knowledge discovery by inference techniques most likely is always the more expensive operation.

[1] An Java API for WSMO Ontologies, avaliable at: http://wsmo4j.sourceforge.net

3.2.2 Object Knowledge Discovery

Within object knowledge discovery, the *compatibility of the objects* defined in the respective WSMO resources is determined. With in the example above for buying and selling an Object O, object knowledge discovery determines whether the “Os” of *B* and *S* are compatible, meaning: when O_B is a chair and O_S is a armchair it should match – but when O_S is table it does not match.

The realization of object knowledge discovery in the course of the SWF project relies on the set based approach of discovery based on simple semantic descriptions defined in [Keller et al., 2004]. As the technical platform for object knowledge discovery, we use the First Order Logic theorem prover VAMPIRE [Riazanov and Voronkov, 2002], version 6.0[2]. The following briefly resumes the theoretical model and explains the realization within VAMPIRE.

3.2.2.1 Matchmaking Notions

The theoretical approach defined in section 4.2.1 of the WSMO Discovery framework identifies four types of matchmaking notions that we apply for object knowledge matchmaking [Keller et al., 2004]. While the theoretical aspects of the different notions are discussed in detail in the referred document, we here resume their definition along with some further insights detected during elaboration of the SWF Discoverers.

3.2.2.1.1 Exact Match

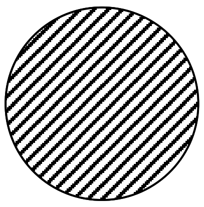
This means that the notions of the request resource and the one of the result resource have to be equivalent. Replacing the predicates g(X) and ws(X) defined in [Keller et al., 2004] by request(X) and result(X) in accordance to the understanding of Discovery Request and Discovery Result defined above, the matching notion in WSML syntax would be:

```
exactMatch(request, result) impliedBy
   forAll X ( request(X) equivalent result(X))
```

What this means is that the notion in the WSMO resource aligned with the Discovery Request have to coincide with the one of the WSMO resource aligned with the Discovery Result, i.e. the have to be exactly the same under consideration of the ontologies used in the resources’ descriptions.

[2] available at: http://www.cs.miami.edu/~tptp/CASC/J2/Systems.tgz

When using this Matchmaking Notion for object knowledge matchmaking, it means that the objects in the description notion WSMO resource aligned with the Discovery Result has to be equivalent to the ones of the resource aligned with the Discovery Request. Spoken more clearly for discovering matching services for a goal: everything required by the goal has to be provided by the web service, but nothing more. Figure 9 shows this correlation.



Legend:

▧ information space of the request resource, i.e. set of all possible instances that would satisfy the description notion

▨ information space of the result resource, i.e. set of all possible instances that would satisfy the description notion

Figure 9: Exact Match Overview

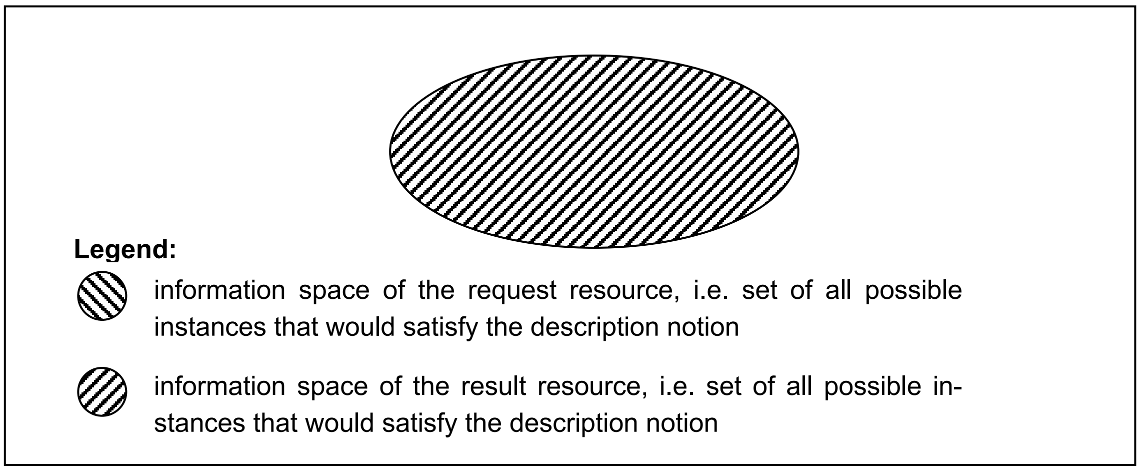
An example for usage of the exact match notion is for matchmaking of effects between Goals and Web Services. It states that for a Web Service to match the Goal, all effects defined in the Goal have to be matching to the effects of the Web Service, but no further effects should be existent in the Web Service description.

3.2.2.1.2 Plugin Match

The plugin match notion specifies that a match is achieved when the notion of the result resource are the same or a superset of the one in the request resource. For the predicate for the result resource being the same or a superset of one of the request resource, the request resource predicate has to logically imply the result resource predicate under consideration of the ontologies used in the resources' descriptions. This matching notion is defined as an implication, written in WSML syntax with the same conventions as above:

```
pluginMatch(request, result) impliedBy
  forAll X ( request(X) implies result(X))
```

When using this Matchmaking Notion for object knowledge matchmaking, it means that the objects in the description notion WSMO resource aligned with the Discovery Result has to be a superset set or be equivalent to the ones of the resource aligned with the Discovery Request. Spoken more clearly for discovering matching services for a goal: everything required by the goal has to be provided by the web service, but also more can be provided but the result resource. Figure 10 clarifies this.



Legend:

▧ information space of the request resource, i.e. set of all possible instances that would satisfy the description notion

▨ information space of the result resource, i.e. set of all possible instances that would satisfy the description notion

Figure 10: Plugin Match Overview

The plugin match can be used within a Discovery Request that shall express that all objects that satisfy the description notion of the request resource have to be provided by the result resource. For instance, the SWF Use Case [Stollberg, 2004] defines goals and services for gathering product information for all products that fulfil certain constraints. another usage example is the example of a goal for finding all Italian restaurants in a city: if there is a service which provides information on all restaurants in a city (where French, Italian, etc. restaurants are subclasses of restaurant), the plugin match is the appropriate matching notion for a Discovery Request that is aligned with the goal.

The realization of the plugin match as a logical implication has some impacts on the conduct for matchmaking. First, if the request resource defines an object X and the result resources defines multiple objects X, Y, Z, then, if the X of both resources is the same, the plugin match holds. This is the obvious and easy part. More interesting is the matchmaking behavior of the implication when the ontological structure of the common object X defined in the request and the result resource is slightly different. We explain our experiences here.

Basically, there are two aspects which result as an impact on the matchmaking behavior when using a logical implication for realization of the plugin match:

(1) **The ontological structure of the object on left hand side of the implication has to be as least as complete as the one of the Object on the right hand side of the implication**

This means that the match only holds if for each attribute defined in the object of the result resource, this attribute also needs to be defined in the corresponding object in the request resource. Table 1 clarifies this.

Table 1: Matchmaking behavior PlugIn Match on Object Structure

matching case		not matching case	
request (left side)	**result (left side)**	**request (left side)**	**result (left side)**
`X memberOf c [ attr. hasValue A, attr. hasValue B, attr. hasValue C, ].`	`X memberOf c [ attr. hasValue A, attr. hasValue C, ].`	`X memberOf c [ attr. hasValue A, attr. hasValue C, ].`	`X memberOf c [ attr. hasValue A, attr. hasValue B, attr. hasValue C, ].`

The reason for this behavior is that an empty, not existing value can logically not imply anything – basic logics.

The background of this issue is the different semantics between intuitive understanding of not defined attribute values within a logical expression of a WSMO resource description in WSML and in logics. In WSML, when the corresponding ontology models a concept X with three attributes A, B, and C, and the object in a resource description is modelled as “O memberOf X” with values only for two attributes A and B, the intended meaning is that their shall be a ‘complete’ instance of X including a value for C, but the attribute value of C is not of interest (i.e. there is no restriction on the values allowed for C). Within logics, a missing attribute value for C in the resource description means that there shall be no value for C.

A solution for this issue is to generate so-called ‘complete facts’[3] out of the resource description; a complete fact for an object in a resource description can be generated by adding existentially quantified variables as values for all attributes defined in the corresponding ontology schema. This allows retrieving a match via the plugin match implication for resources whose objects have incomplete and different ontological structures

[3] the term ‘complete fact’ has been introduced within the discussions on discovery realization in the WSMO working group. It means that an object in a resource description is described with attributes for all attributes defined in the concept definition of the corresponding domain ontology.

with regard to the attribute values specified explicitly in the WSML resource description.

(2) **The implication only holds if the left hand side is more specific than the right hand side**

Apart from the 'completeness of object descriptions' discussed above, a second aspect results from the realization of the plugin match as a logical implication concerned with the generality of attribute values defined on the left hand side and the right hand side of the implication. Referring to the SWF Use Case, the following domain ontology knowledge is defined: 'seller' is superconcept of 'company', and IKEA is an instance of 'company'. Herefore, the following holds:

```
[1] IKEA implies (exists ?X(?X memberOf company) )      TRUE

[2] ?X(?X memberOf company) implies IKEA                FALSE
```

The reason for this is also basic logics, as only a more general term is implied by a more specific term (as in formula [1]), but not the other way around (as in formula [2]).

This holds in general – and there is no way (and there should not be one) to work around this. In combination with the aspect on 'completeness of object description', we can summarize two rules for the matchmaking behavior of the plugin match:

(1) the implication only holds if the ontological structure of the object defined in the request resource (left hand side) is as least as complete as the ontological structure of the object defined in the result resource (right hand side); when generating complete facts for the object descriptions, this rule gets obsolete.

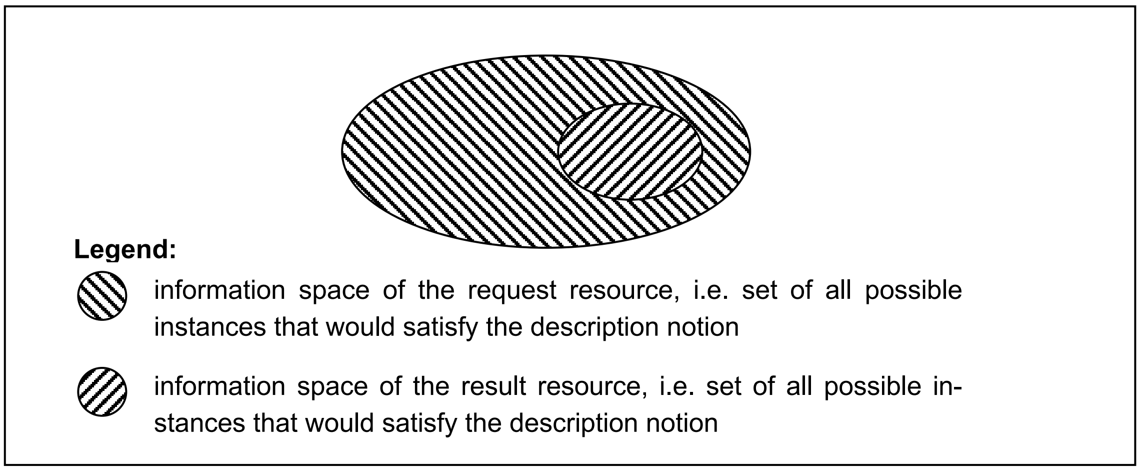
(2) *the implication only holds if for each attribute defined in the object descriptions of the request and the result resource holds that the attribute value of the object in the request resource (left hand side) has to be more specific than the one of the object in the result resource (right hand side).*

3.2.2.1.3 Subsumption Match

The subsumption matching notion is the inversion of the plugin match: it specifies that a match is achieved when the notion of the request resource are the same or a superset of the one in the result resource. The subsumption match is realized as an inverse logical implication, written in WSML syntax:

```
    subsumptionMatch(request, result) impliedBy
      forAll X ( request(X) impliedBy result(X))
```

Due to its definition, the matchmaking behavior of the subsumption match is inverse to the one of the plugin match. There is a match if the objects in the request resource are a superset set or are equivalent to the ones of the result resource, as shown in Figure 11.



Legend:

▧ information space of the request resource, i.e. set of all possible instances that would satisfy the description notion

▨ information space of the result resource, i.e. set of all possible instances that would satisfy the description notion

Figure 11: Subsumption Match Overview

Although the subsumption match is not applied within the discovery mechanisms realized for the SWF Use Case prototype, we deduce the applicability of this matching notion for discovering result resources that have to be composed in order to match the objects of the request resource.

Considering the Web Service Discovery for a Goal, using the subsumption matching notion means that if there is a Goal G which specifies three objects X, Y, and Z to be achieved, and there is a Web Service WS1 which specifies a object X, then G and WS1 match according to the subsumption match. In order to detect Web Services that can resolve G completely, some further Web Services WS2 … WSn need to be used that provide results match the objects Y and Z. When discovering these Web Services with the subsumption match, you can infer that the discovered Web Services need to be composed in order to resolve Goal G.

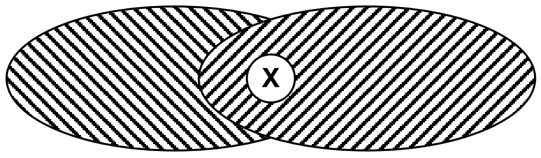
Because of the realization of the subsumption match as a logical implication, the same aspects identified for the plugin match hold here as well in an inverse manner.

3.2.2.1.4 Intersection Match

The intersection match notion defines a match when there is an object which holds for the object defined in the request resource as well as for the object defined in the result resource. Thus, in contrast to the matching notions explained above, the intersection match is a existentially quantified notion – in WSML syntax:

```
intersectionMatch(request, result) impliedBy
  exists X ( request(X) and result(X))
```

This means that if there is one object existing that satisfies the constraints defined in the description predicate of the request resource as well as those of the result resource, then these resources match. Figure 12 clarifies this.



Legend:

▧ information space of the request resource, i.e. set of all possible instances that would satisfy the description notion

▨ information space of the result resource, i.e. set of all possible instances that would satisfy the description notion

Ⓧ existing instance that satisfies object in the description notion of request resource as well as the one of the result resource

Figure 12: Intersection Match Overview

Regarding the realization of the intersection match, there needs to be an existing object that satisfies object in the description notion of request resource as well as the one of the result resource. In fact, satisfying the object description means that there is no contraction between the object in the request resource and the result resource. We explain the technical realization in VAMPIRE below (section 3.2.2.2).

Applying the intersection match notion within a Discovery Request means that the Discovery Result shall be comprised of all resources that can provide one object that satisfies the restriction defined in the object description of the request resource. An example from the SWF Use Case are Goals and Services that are concerned with purchasing a product. A Goal that specifies "buy 1 product of type X" should be matching with a Web Service that specifies "sell product of type X" (note: the required matching is different in SWF due to the overall system design). Whenever the intended meaning of the Discovery Request is to receive one object wherefore the constraints defined in the object description hold, the intersection match is the appropriate matching notion.

3.2.2.2 Realization with VAMPIRE

VAMPIRE is a resolution based system for completely automatic theorem proving in first-order classical logic with equality, developed by Andrei Voronkov and Alexandre Riazanov at the Computer Science Department, University of Manchester. VAMPIRE has been competing in the CADE Automated Theorem Proving System Competition (CASC), a competition on theorem proving system that is yearly performed in conjunction with the International Conference on Automated Deduction (CADE). Out of several existing systems we have chosen VAMPIRE because of its excellent performance in the CASC competition over the last years.[4]

3.2.2.2.1 Knowledge Representation

VAMPIRE works on TPTP syntax, a First Order Logic syntax which is used as a common syntax within the CASC competition. The transformation from WSML syntax to TPTP can be achieved without loss of information, because TPTP as well as WSML are full FOL languages.[5] Without referring to syntactical issues, we explain which knowledge representations are needed within Vampire and how they are derived from WSML.

(1) Ontological Theory

VAMPIRE is a basic tool without any built in functions. Every knowledge that shall be used as a basis for calculations has to be provided as a FOL theory. So, to support ontological reasoning, we have to define an ontological theory.

In order to support ontological reasoning on WSMO ontologies, the WSMO meta model for ontologies has to be transformed into a FOL theory. This theory consists of ontology schema descriptions as defined in [Roman et al., 2004]. In addition, the basic ontology relations that are not supported within FOL have to defined; for VAMPIRE, these are transitive subsumption of instances and attributes. [6]

[4] For further information on VAMPIRE and related aspects we refer to the VAMPIRE homepage at: http://www.cs.man.ac.uk/~riazanoa/Vampire/

[5] for TPTP syntax specification and related information see: http://www.cs.miami.edu/~tptp/. The different WSML specifies are under construction at the time of writing, we refer to [de Bruijn, 2004]. The transformation from WSML to TPTP is supported by the WSML Parser: from a WSML specification, an abstract syntax tree is derived which is then represented in a intermediary FOL syntax; thereof, any FOL compliant syntax can be derived as needed for a the technical environment used. The WSML Parser builds upon the WSML Validator (http://dev1.deri.at:8080/wsml/), and is under construction at the time of writing.

[6] For the realization of the SWF Use Case, we have defined a preliminary version of the WSMO ontology theory available at: http://cvs.deri.at/cgi-bin/viewcvs.cgi/swf/resources/ontologies/onto.ax

(2) Universe Definition

The universe holds the ontology schema definitions as well as so-called ‘generic instances’ of the application specific domain ontologies. While the ontology schema definitions are needed to allow ontological reasoning within the resources description that use the domain ontologies, the ‘generic instances’ are needed for realization of the Intersection Match.

A ‘generic instance’ is defined like follows. For a concept X defined in a WSMO domain ontology that has three attributes A, B, and C, a generic instance is defined as the existential existence of an instance wherefore each attribute has a universally quantified variable as its attribute value. Written in WSML syntax:

```
forAll ?A, ?B, ?C(
      exists ?X(?X instanceOf concept[
            attributeA hasValue ?A,
            attributeB hasValue ?B,
            attributeC hasValue ?C
            ]
      )
 ).
```

Listing 1: Universe Definition of a Generic Instance

By defining such a construct for every concept that is defined in the ontology schema of the domain ontologies used within an application, a generic instance for every possible ontology object that can be defined in a WSMO resource description can be constructed. For example, if the attribute A of concept X in the Listing above is of type T, and there exists a generic instance definition for a concept Y which is instanceOf type T, a graph can be determined that represents a complete, generic instance of X. The generic instances for the universe definition of an application can be automatically derived from the domain ontology definitions.

The definition of is the prerequisite for the intersection match as explained above (see 3.2.2.1.4). VAMPIRE has to find an existing instance of the object X, which can be a complex ontology construct. With the existing generic instances defined for the domain ontology schemas, there always exists a generic instances for all ontology constructs that are valid according to the schema definitions in the used domain ontologies. On this basis, VAMPIRE proves whether the constraints defined in the object definition of the request resource and the result resource do not contradict in order to determine the matching of the resources using the Intersection Matchmaking notion.[7]

The approach of defining a universe as described here is not new, but complies to related work for realization of the Intersection Match. The related work will be discussed elsewhere, as this document only presents an overview of the realization of WSMO Discovery in SWF.

(3) Knowledge Base

Not mandatory, you can define a Knowledge Base for your application – which then of course also has to be provided to the technical platform for object knowledge discovery.

A Knowledge Base consists of instance definitions (and only of instance definition – all other ontology knowledge needed for the application is provided by the universe definition).[8] These instances are pre-defined instances that are used within object definitions in resource descriptions; for example, when pre-defining 'innsbruck' as an instance of 'city' in the knowledge base, you can simply use this instance in attribute value definitions within an object definition. Within the elaboration of the SWF Use Case, we detected the definition of such a knowledge base very convenient for preventing huge and unreadable object definitions.

The instance definition can be automatically transformed from WSML definitions to TPTP or any other backend syntax needed.

(4) Resource Descriptions

The final knowledge needed for object matchmaking with VAMPIRE are the object description of the request and result resources. For all resources, only the logical formulas of the relevant object descriptions are needed.

While the relevant aspects of a WSML description of a resources can be accessed via the WSMO4J API, the logical expressions can be translated with the WSMO Parser. As both tools are under construction at the point of writing, the resources of the SWF Use Case have been transformed automatically.

[7] The universe definition for the SWF Use Case is available at: http://cvs.deri.at/cgi-bin/viewcvs.cgi/swf/resources/ontologies/universe.ax. Please note that this universe definition has been created manually since the technology for transformation from WSML to TPTP and for automated generation of universe definitions has not been available at the time of creation.

[8] The ontology schema definitions are moved into the universe definition as mandatory domain knowledge.

3.2.2.2.2 Proof Obligation

VAMPRIE allows to include theories, i.e. files that contain theories in TPTP format. Thus, the proof obligation to be sent as input to the prover can be created dynamically. The general structure of such a proof obligation consists of three parts:

1) include files with ontology knowledge
2) include relevant object for resources to be matched
3) include the Matching Notion to be applied

This structure allows usage of VAMPIRE with dynamically created proof obligations, as we have realized within the GG and GS Discoverer.

3.2.2.2.3 Strengths and Weaknesses of VAMPIRE

The strength of VAMPIRE, and of theorem provers in general in comparison to reasoners that work on FOL derivates like FLORA2 and OntoBroker for F-Logic, or Description Logic Reasoners like FaCT or RAZOR, is that knowledge items do not have to be inserted into the system's internal knowledge base for matchmaking. While reasoners usually determine the satisfaction of a rule body by applying knowledge that is available in the internal knowledge base, a theorem prover verifies a proof obligation by determining the provability on basis of the structure of logical formulas.

For example, the approach for realization of discovery with FLORA2 presented in [Kifer et al., 2004] needs to hypothetically insert knowledge into the internal knowledge base for determining the satisfaction of logical formulas, e.g. in web service postconditions. Within VAMPIRE, we only need to define the universe knowledge at system setup time (and optionally a knowledge base, as outlined above), while no further knowledge needs to be created or stored intermediary for determining the proof obligations at run time. Besides, with TPTP a FOL language is supported so that possible occurring problems of expressiveness between WSML (Full) and the system language are debarred.

Concerning stability and system performance, VAMPIRE requires a lot of memory and is not capable to process multiple thread simultaneously. This holds in general for inference machines that have been developed within academic efforts. VAMPIRE runs stable (at least we did not notice any system crashes in our testing phase), and the proofs have been found within a range of 0,1 up to 2 seconds (which of course is related to the simplicity of the proof obligations – with more complex formulas, the runtime increased significantly; it is possible to set a time limit for proving, so that system hang-ups can be avoided). Compared to other theorem provers, the performance of VAMPIRE has been outranging: a proof obligation that took less than a second in VAMPIRE required nearly 2 minutes computation time in OTTER.

A major weakness of VAMPIRE is that it does not provide any built-in functions for processing basic datatypes and arithmetics; only FOL formulas with variables and constants can be processed. All knowledge beyond this has to be defined explicitly in theories. This obviously hampers the applicability of VAMPIRE, as features like string comparison or arithmetic calculations are not supported by default.

3.3 Modeling of Resources

The extended framework for discovery introduced above requires a specific way of modeling the WSMO resources.

3.3.1 Action Knowledge Description

Action Knowledge defines which resources are compatible with regard to the actions, but without regard to the objects action defined within a resource. It is described in additional ontologies with a special structure, which we refer to as *action knowledge ontologies.* These ontologies describe the action knowledge for all resources used in an application and define the compatibility of resources according to the actions.

The general structure of an action knowledge ontology is determined by a taxonomy of actions and a taxonomy of resources (that are the WSMO resources). An Action has a set-valued attribute 'compatibleAction' that defines the compatible actions, and a resource has a set-valued attribute 'hasAction' that defines the actions that the resource. Another way of expressing compatibility of actions would be to define a relation 'actionCompatibility' that holds the compatible actions in parameters, but no major benefit can be achieved by this.

Listing 2 shows a simple example of an action knowledge ontology with the compatible actions 'buy' and 'sell', and corresponding instances of resource, a Goal and a Service.

```
concept action
  compatibleAction ofType set action

concept buy subConceptOf action
  compatibleAction ofType set sell

concept sell subConceptOf action
  compatibleAction ofType set buy
```

```
concept resource
  hasAction ofType set action

concept goal subConceptOf resource

concept service subConceptOf resource


concept buyergoal subConceptOf goal
   hasAction ofType set buy

concept sellerservice subConceptOf service
   hasAction ofType set sell


instance <<http://.../goal1>> memberOf buyergoal

instance <<http://.../ws7>> memberOf sellerservice
```

Listing 2: Example Action Ontology

The example shows a ‘binary action compatibility’, meaning that only 2 resources have to interact; ‘n-ary action compatibility’ is also supported by the attribute ‘compatibleAction’ being set-valued. For ‘n-ary action compatibility’, the problem is to define which resources have to interact in what way. For instance, an action ‘purchase for buyer’ shall be compatible to ‘sell’ and ‘deliver’; this can be understood as a binary action compatibility wherein the actions ‘sell’ and ‘deliver’ have to be combined into an action ‘purchase for seller’ which then is declared to be compatible to ‘purchase for buyer’. This refers to the more general question of who to describe n-ary interactions between resources. Within the elaboration of the SWF discoverers, we restricted ourselves to solutions for binary interactions.

The definition of the action knowledge is related to design aspects of an application. The action knowledge is implicitly defined within the domain ontologies, (or rather: the domain ontologies are mostly designed with having the action items for the application in mind); as well the resources in an application are mostly designed and described with having a model of compatibility and interaction in mind that refers to what we define here as action knowledge. As we will show below in examples, the action knowledge and its use within discovery seems to be necessary since the matchmaking between logical expressions in resource might return false discovery results, even if the resources are described ‘correctly’ from the application point of view.

3.3.2 Object Description in WSMO Resources

To support the application of the matchmaking notions for object matching described above, a specific way of modelling the logical expressions in resource description is required. The reason is that the predicate structure as defined in the matching notion, where an object X is required to be defined. Therefore, we model logical expressions in resources as shown in Listing 3, which is valid WSML[9].

A free, i.e. not quantified variable X defines the object that is desired. Other variables that are needed in the logical expression are quantified explicitly; each variable has to be typed. With this structure, a logical expression of a description notion in a resource defines the predicate that is applied within the matchmaking notions (see 3.2.2.1).

```
effect
 axiom bgs1Effect
    nonFunctionalProperties
        dc:description hasValue "direct delivery of the purchased piece of furniture to a
            specific shipping address in Austria"
    endNonFunctionalProperties
    definedBy
        exists ?Item(?Item memberOf swfmo:product) and
        exists ?Buyer(?Buyer[
             shipToAddress hasValue ?Address
        ] memberOf swfmo:buyer) and
        exists ?Address(?Address[
            city.inCountry hasValue austria
        ] memberOf loc:address) and
        ?X[
             deliveryItem hasValue ?Item,
             receiver hasValue ?Buyer
        ] memberOf swfmo:dropShip .
```

Listing 3: Example logical expression[10]

For modeling predicate that has several ontology objects as the object to be expressed in a logical formula, the free variable X can be combined by con- or disjunctions of ontology objects.

3.4 Conclusion

We have outlined one possible approach to realizing the framework of WSMO Discovery as defined in [Keller et al., 2004]. Our approach works on the specification of

[9] SWF uses valid WSML according to WSMO D2, that can be verified with the WSML Validator, version 1.0: http://dev1.deri.at:8080/wsml/

[10] taken from goal definition for Buyer Goal Template 1 [Stollberg, 2004].

WSMO resources as defined in WSMO version 1.0 [Roman et al.,2004]. We do not claim the approach outlined above to be the only possible one, but it satisfies the needs for discovery in the SWF project.

Some aspects can be adhered in general for realization of the WSMO Discovery framework:

1) The WSMO Discovery framework is not restricted to Web Service Discovery, i.e. detection of (Web) Services that possibly can solve a Goal. Therefore, we defined the more general notions of Discovery Request and Discovery Result that can be aligned with any WSMO top level component.

   An example of a discoverer different from Web Service discovery is the GG Discoverer in SWF that detects compatible Goals (see section 4.2).

2) In order to provide a discoverer with sufficient knowledge needed to determine high quality discovery results, we extend the notion of a Discovery Request with additional, more specified input information. We distinguish between Action Knowledge and Object Knowledge for a resource description; also the required matchmaking notion is part of a Discovery Request as the applicability of the different matchmaking notion is determined by the actual desire that is to be expressed.

3) We provided insights on the usage of this different matchmaking notions for object discovery that resulted from our work.

4) The usage of the FOL theorem prover VAMPIRE seems to have advantages for realization of object matchmaking, as intermediary insertions of knowledge items into the internal system’s knowledge base get obsolete. With TPTP, a FOL language is supported that allows transformation of WSML (Full) expressions without loss of information.

   Nevertheless, due to lack to processing support for basic datatypes and arithmetics and the maturity as a software tool, VAMPIRE can not be considered as a core building block for advanced information processing systems.

5) The extended framework requires specific guidelines for modeling of knowledge items and resources. We assume that such guidelines are required for realization of discovery independent of the backend tool used.

4 SWF Discoverers

After specifying the technical architecture for SWF, this section specifies the SWF Discovery Mechanisms. We explain the concepts for discovery, and specify the technical realization of each discoverer. For examples within the discoverers we refer to the SWF Use Case defined in [Stollberg, 2004].

This section is organized as follows: 4.1 provides an overview of the implementation of the SWF Discoverers. The subsequent sections specify the distinct SWF Discovery mechanisms: 4.2 (GG Discoverer), 4.3 (GS Discoverer), and 4.4 (WW Discoverer).

4.1 Implementation Overview

This section explains the implementation concept of the SWF Discoverers as main components of the SWF architecture outlined in section 2.

4.1.1 Discovery Manager

The SWF Framework identifies three subsequently working discovery mechanisms for establishing a cooperation between Fred-agent: GG Discovery detects potential cooperation partners on basis of the compatibility of Goals assigned to Freds, GS Discovery discoverers possibly usable services for each cooperation partner, and WW Discovery determines the compatibility of the services used by potential cooperation partners with regard to the services' choreographies. After successful cooperation establishment, i.e. finalization of these mechanisms, the Freds along with their related resources are sent into a meeting of executing the cooperation [Stollberg et al., 2004].

Obviously, the three mechanisms for cooperation establishment have a sequential dependency – the GS Discovery works on the result of the GG Discovery, and the WW Discovery on the result of the GS Discoverer. For decoupling of the distinct discoverer functionalities, we specify the SWF Discovery Manager that stores the discovery results of the separate discoverers, invokes each discoverer on basis of a dynamically created, intermediate discovery result of the preceding discoverer, and finally determines a cooperation as the overall result of the cooperation establishment mechanisms. The SWF Discovery Manager is located in the Goal Solver that control the resolution process of Goal Instances that are assigned to Freds for automated resolution (see section 2.2).

In overall cooperation establishment process shown in Figure 13 commences with a Goal Instance, i.e. a concrete task assigned to a Fred for automated resolution. For this

Goal Instance, which is referred to as the **initiating Goal Instance** $\mathtt{GI}_i$, all possible co-operations are determined by finding potential cooperation partners (GG Discovery), possibly usable services for each partner (GS Discovery) that are compatible with regard to their consumption interface (WW Discovery). A cooperation established via the discovery mechanisms consists of:

- 1 up to n **Freds** (the Freds are the electronic representatives of cooperation partners; a cooperation with only 1 Fred can be established for Goals that do not need cooperative goal resolution; this is called a singleton meeting)
- **Goal Instances** of the Freds (each participating Fred has 1 Goal Instance that is to be solved by the cooperation)
- **Services** (1 up to n services for each cooperation partner; when an error occurs during service execution in a meeting, a different service can be used so that the complete cooperation does not have to be cancelled)
- Needed resources:
  - **Ontologies** used in Goal and Service Description
  - **Mediators** used in Goal and Service Description, and those invoked within some Discoverer (this is only conceptual, not realized in SWF)

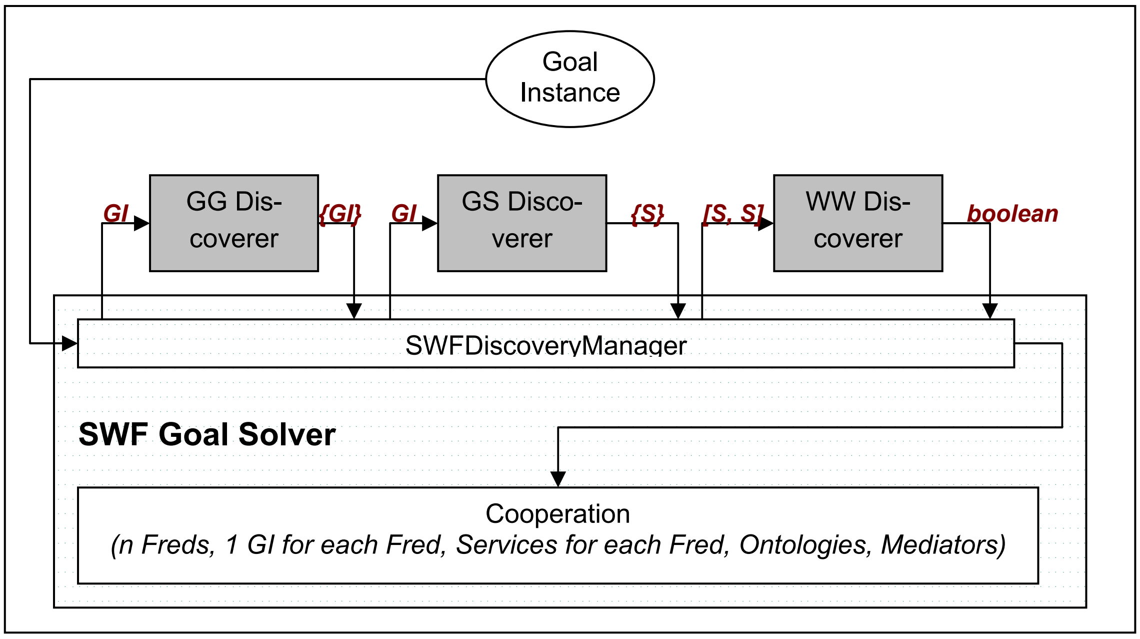


Figure 13: SWF Discovery Manager Overview

The SWF Discovery Manager receives the initiating Goal Instance GI_i, invokes the separate discoverers and collects their discovery results in intermediary cooperation establishment results. Figure 14 shows an overview of the SWF Discovery Manager workflow with further explanations below.

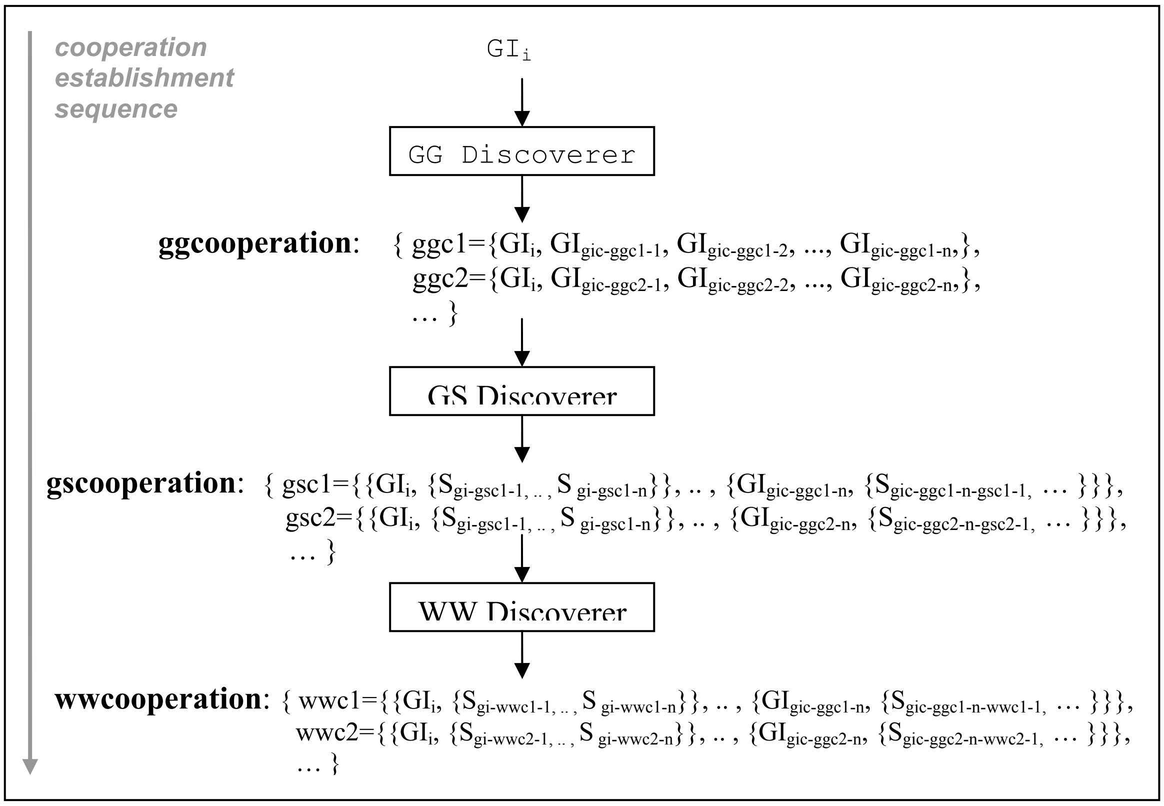

cooperation establishment sequence
GIi
GG Discoverer
ggcooperation: { ggc1={GIi, GIgic-ggc1-1, GIgic-ggc1-2, ..., GIgic-ggc1-n,}, ggc2={GIi, GIgic-ggc2-1, GIgic-ggc2-2, ..., GIgic-ggc2-n,}, … }
GS Discoverer
gscooperation: { gsc1={{GIi, {Sgi-gsc1-1, .. , S gi-gsc1-n}}, .. , {GIgic-ggc1-n, {Sgic-ggc1-n-gsc1-1, … }}}, gsc2={{GIi, {Sgi-gsc1-1, .. , S gi-gsc1-n}}, .. , {GIgic-ggc2-n, {Sgic-ggc2-n-gsc2-1, … }}}, … }
WW Discoverer
wwcooperation: { wwc1={{GIi, {Sgi-wwc1-1, .. , S gi-wwc1-n}}, .. , {GIgic-ggc1-n, {Sgic-ggc1-n-wwc1-1, … }}}, wwc2={{GIi, {Sgi-wwc2-1, .. , S gi-wwc2-n}}, .. , {GIgic-ggc2-n, {Sgic-ggc2-n-wwc2-1, … }}}, … }

Figure 14: Intermediate Results for Cooperation Establishment

4.1.1.1 GG Cooperation

A ggcooperation gathers the discovery result of the GG Discoverer. For the initiating Goal Instance GI_i, all compatible Goal Instances with the processing status ‘open’ (i.e. to be solved) are detected. A compatible Goal Instance with the status ‘open’ is defined as a task assigned to a Fred for automated resolution, so that represents a set of compatible Goal represents a possible cooperation of Freds as a potential cooperation partners.

The overall ggcooperation can consist of several sub-ggcooperations, denoted as ggc1 … ggcn. Each sub-ggcooperation represents one set of potential cooperation partners; for example, if GI_i can be solved by cooperating with GI_A, but also when cooperating with GI_B, there are two sub-ggcooperation ggc1={GI_i, GI_A} and ggc1={GI_i, GI_B}. Each sub-ggcooperation can consist of 0 .. n compatible Goal Instances, denoting the arity of the ggcooperation ("0" means a singleton cooperation, "n" a n-ary cooperation). The Goal Instances in a sub-ggcooperation are denoted by 3 indices: "gic" means that it is compatible to the initiating Goal Instance GI_i; the second index indicates the sub-ggcooperation the Goal Instance belongs to; the last index enumerates the Goal Instances in the sub-ggcooperation, so that the highest number indicates the arity of the sub-ggcooperation.

4.1.1.2 GS Cooperation

A gscooperation gathers the discovery result of the GS Discoverer which detects possibly usable services for each Goal Instance in every sub-ggcooperation of the overall ggcooperation separately.

The general structure of a gscooperation is that each Goal Instance of the ggcooperation is assigned a set of Services. The indices for the Services denote the Goal Instance and the sub-gscooperation that the service belongs, following the same nomenclature as explained above for ggcooperations.

In principle, a gscooperation represents the corresponding ggcooperation, adding a set of possibly usable services to each Goal Instance as the result of GG Discovery.

4.1.1.3 WW Cooperation

A wwcooperation gathers the WW Discovery Result. The WW Discoverer takes all possible combinations of services of all sub-gscooperations and determines their compatibility with regard to the services' choreography. When a service is not compatible in this respect, it is removed from the set of services in its respective sub-gscooperation.

An example for clarifying the functionality of WW Discovery. There is a sub-gscooperation with Goal Instance GI_i, and a Goal Instance GI_A; GS Discovery has detected 3 possibly usable services for GI_i: S_{gi-1}, S_{gi-2}, S_{gi-3}, and 2 services for GI_A: S_{gA-1}, S_{gA-2}. The service choreography compatibility check in the WW Discoverer determines the following pairs to be compatible: $\{S_{gi-1}, S_{gA-1}\}$, $\{S_{gi-1}, S_{gA-2}\}$, $\{S_{gi-3}, S_{gA-2}\}$. So, the service S_{gi-2} is not compatible to any service of GI_A, and thus is removed from the set of services for GI_i in the sub-wwcooperation.

In principle, a wwcooperation represents the corresponding gscooperation with a reduced set of services for each Goal Instance with respect to the compatibility of the services' choreography compatibility.

4.1.2 SWF Discoverer Modules

This section provides a brief overview of the technical realization of the SWF Discovery Modules developed by DERI. The modules are developed in Java, JDK 1.4, and are available as completely open source at the SWF CVS.[11]

4.1.2.1 Discoverer Interfaces

We define Java Interface that reflect the general architecture of discoverers as outlined in section 3.1. The specific SWF Discoverer implement these interfaces.

Table 2: SWF Discovery Java Interfaces

Name	main methods
IDiscoverer	discover (IDiscoveryRequest), returns IDiscoveryRequest
IDiscoveryRequest	getter / setter for WSMO top level notion and Registry to be used
IDiscoveryResult	getter / setter for WSMO top level notion, for related IDiscoveryRequest, and information about discovery process
IIdentifier	getter for Identifier (supports different types of identifiers – URL, WSML-identifiers, Object-identifiers)
IRegistry	getter for Registry and WSMO resource types stored in registry, store & get WSML descriptions
IRegistryDirectory	management of several registries
IRegistryRestriction	acceptance & restrictions for a IRegistry
IProverConnector	check(proofobligation), invoking / connecting to a prover for object matchmaking

4.1.2.2 Package Structure

The following gives an overview of the Java package / CVS structure of the SWF Discoverer implementation. All sources are available in the SWF CVS (web interface see footnote 11)

[11] SWF CVS web interface: http://cvs.deri.at/cgi-bin/viewcvs.cgi/swf/

Table 3: SWF Java Package and CVS structure

Name	main methods
/bin/	has the VAMPIRE binaries
/src/	contains the Java resources for the discoverer modules
src/org/deri/swf/discovery/gg/ ValidatingGGDiscoverer.java	GGDiscoverer Class
src/org/deri/swf/discovery/gs/ GSDiscoverer.java	GSDiscoverer Class
src/org/deri/swf/discovery/ww/	WWDiscoverer Class (only dummy currently)
src/org/deri/swf/prover/	all Java classes needed for connecting / calling Vampire in the discoverers
src/org/deri/wsml/parser/	contains the Parser from WSML to TPTP (not existing at time of writing)
src/org/deri/wsmo/discovery/api/	contains the Java Discovery Interfaces
src/org/deri/wsmo/registry/	Registry handling classes

4.1.2.3 VAMPIRE as Web Service

We use VAMPIRE version 6.0, which has won the CASC competition in 2000 (see section 3.2.2.2). Although there is a newer version available, we chose version 6.0 as its supports a specific operation mode which allows proof making significantly faster.

VAMPIRE runs under Linux / UNIX. In order to allow usage within the Windows-based SWF platform, and in order to make it available for other applications, we established a (not semantic) Web Service that implements the IProverConnector Interface. It is available at http://dev1.deri.at:8080/vampire/services, with the WSDL Interface shown in Listing 4. We use AXIS version as the Web Service platform in conjunction with Tomcat 5 on a Red Hat Linux server.

```
<?xml version="1.0" encoding="UTF-8" ?>
  <wsdl:definitions targetName-
    space="http://138.232.65.151:8080/vampire/services/VampireInvoker"
    xmlns:apachesoap="http://xml.apache.org/xml-soap"
    xmlns:impl="http://138.232.65.151:8080/vampire/services/VampireInvoker"
    xmlns:intf="http://138.232.65.151:8080/vampire/services/VampireInvoker"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
- <!--
WSDL created by Apache Axis version: 1.2beta
Built on Mar 31, 2004 (12:47:03 EST)
  -->
<wsdl:message name="checkResponse">
  <wsdl:part name="checkReturn" type="xsd:boolean" />
  </wsdl:message>
<wsdl:message name="checkRequest">
  <wsdl:part name="po_content" type="soapenc:string" />
  <wsdl:part name="po_id" type="soapenc:string" />
  </wsdl:message>
<wsdl:portType name="LocalProverInvoker">
<wsdl:operation name="check" parameterOrder="po_content po_id">
  <wsdl:input message="impl:checkRequest" name="checkRequest" />
  <wsdl:output message="impl:checkResponse" name="checkResponse" />
  </wsdl:operation>
  </wsdl:portType>
<wsdl:binding name="VampireInvokerSoapBinding" type="impl:LocalProverInvoker">
  <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
<wsdl:operation name="check">
  <wsdlsoap:operation soapAction="" />
<wsdl:input name="checkRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" name-
    space="http://prover.swf.deri.org" use="encoded" />
  </wsdl:input>
<wsdl:output name="checkResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" name-
    space="http://138.232.65.151:8080/vampire/services/VampireInvoker" use="encoded"
    />
  </wsdl:output>
  </wsdl:operation>
  </wsdl:binding>
<wsdl:service name="LocalProverInvokerService">
<wsdl:port binding="impl:VampireInvokerSoapBinding" name="VampireInvoker">
  <wsdlsoap:address loca-
    tion="http://138.232.65.151:8080/vampire/services/VampireInvoker" />
  </wsdl:port>
  </wsdl:service>
  </wsdl:definitions>
```

Listing 4: WSDL Interface of VAMPIRE Web Service

4.2 GG Discoverer

As the first step in the cooperative goal resolution, GG Discovery detects potential cooperation partners by determining the compatibility of their respective Goal Instances. The compatibility is given when the objects of interest (i.e. the Goal postconditions) match, and when the cooperation roles of the Goal Instance owners are compatible.

4.2.1 Overview

The following defines the GG Discoverer as a specialization of the general WSMO discoverer architecture defined in section 3.1.

4.2.1.1 Discovery Request

Assigned WSMO Resource: Goal Instance

This is the Goal Instance of the initiating Fred; a newly created Goal Instance (status: “open) is discovered by selection engines in the FRED-part of SWF.

Action Knowledge: Cooperative Knowledge Ontology

This ontology defines cooperative Goal Templates

Object Knowledge: WSMO description of Goal Instance

Matching Notion: depending on assigned Goal Instance

4.2.1.2 Discovery Result

Assigned WSMO Resource: Goal Instances

A set of compatible Goal Instances is returned. These are Goal Instances (status = open or not resolved) which are owned by potential cooperation partners. Thus, the Discovery Result of GG Discovery represents groups of Freds as potential cooperation partners.

4.2.2 Functionality

Figure 15 shows the overview of the GG Discoverer. The DiscoveryRequest GI (Initiator) is `instanceOf` a Goal Template; this Goal Template belongs to a `compatibleGoalGroup` as defined in the Cooperative Knowledge Ontology. For those Goal Instances which are of Goal Templates compatible to the one of the Initiator (determined by usage of *Cooperative Goal Knowledge*), the GG Matcher determines whether the objects of interest match (object matchmaking with theorem prover).

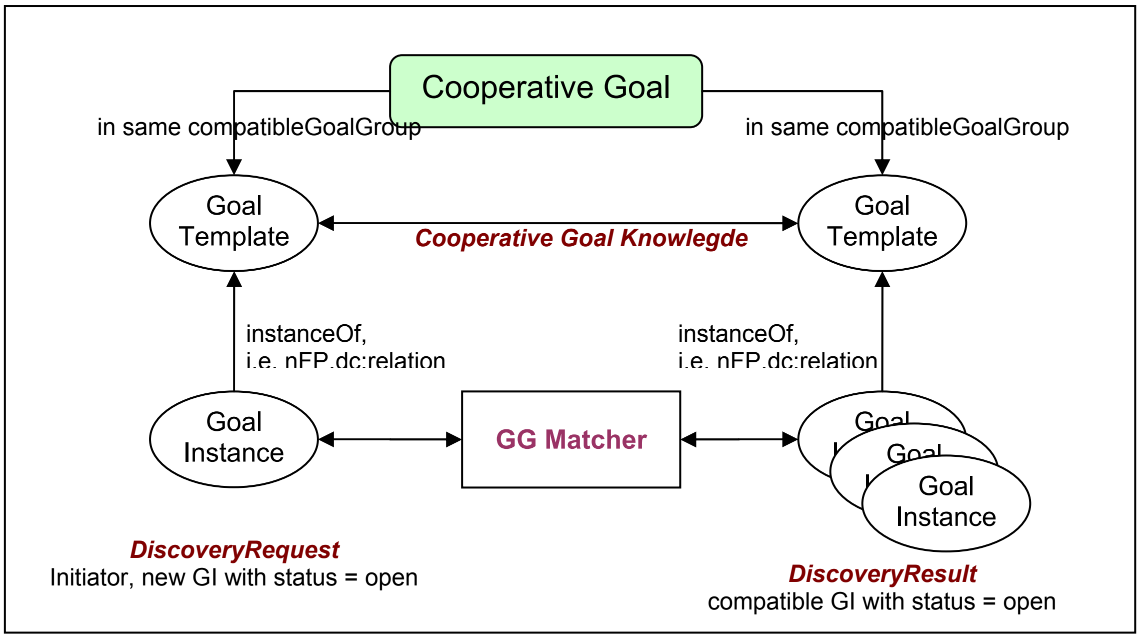

Cooperative Goal
in same compatibleGoalGroup
in same compatibleGoalGroup
Goal Template
Cooperative Goal Knowlegde
Goal Template
instanceOf, i.e. nFP dc:relation
instanceOf, i.e. nFP dc:relation
Goal Instance
GG Matcher
Goal Instance
DiscoveryRequest
Initiator, new GI with status = open
DiscoveryResult
compatible GI with status = open

Figure 15: GG Discoverer Overview

The DiscoveryResult is a set of Goal Instances; each represents a potential collaboration partner for cooperative goal resolution. The GGDiscoveryResult is a ggcooperation kept and handled by the SWF Discovery Manager (see section 4.1.1).

4.2.3 Discovery Mechanisms Specification

The following explains the realization of the GG Discoverer in more detail. Here, we describe the realization of the SWF prototype as presented in a demonstration at the ISWC 2004.[12]

4.2.3.1 Cooperation Knowledge Filter

The Action Knowledge needed for GG Discovery – terminology according to the knowledge type distinction explained in section 3.2.1 – is the compatibility of Goals with regard to the actions defined in Goals. Due to the definition of the notions of Goal Templates and Goal Instances in SWF, we define the compatibility on the Goal Template level and therewith filter Goal Instances that belong to Goal Templates that are compatible to the Goal Template that the initiating Goal Instance is an instance of. This filter is located as the first mechanism in the GG Discoverer, providing a pre-selection

[12] accepted in the Demonstration Session of the 3rd International Semantic Web Conference, ISWC 2004; see: http://iswc2004.semanticweb.org/demos/index.html.

of the Goal Instances that are checked for object matchmaking in the GG Matcher (see below).

A SWF Goal Template defines the structure of goals that can be solved with the services available in the system; in the description notions for postcondition and effects, it defines the general object structure without concrete attribute values. A SWF Goal Instance expresses a concrete desire that is assigned to a Fred for automated resolution. The relation between a SWF Goal Template `GT` and a SWF Goal Instance `GI` is that: `GI` **`instanceOf`** `GT`. This means that for all those description notions `N` that are the same in a `GT` and a `GI` description (which are 'importedOntologies', 'usedMediators', 'postcondition', and 'effect') holds that `N` in the `GI` description is inherited from `N` in the `GT` description; `N` can be refined but not extended. For the notions of postcondition and effect – which describe the desire in logical expressions – this means that `N` in a `GI` can restrict the allowed ontology notions according to the taxonomy of the domain ontologies (i.e. restricting to sub-notions of the ontology object specified in the `N` of the corresponding `GT`, or restricting by defining concrete values for attribute values that are valid for the `N` of the corresponding `GT`). In other words: Goal Instances can only differ from their corresponding Goal Template by refined object knowledge.

By this definition we can ensure that if a GT_i that is the Goal Template that the initiating Goal Instance GI_i is an instance of, then for all Goal Instances GI_{ci} that are instances of a Goal Template GT_{ci} that is compatible to GT_i, then GI_{ci} is compatible to GI_i. More formally:

$(GI_{ci}$ **instanceOf** $GT_{ci})$ **and** $(GI_i$ **instanceOf** $GT_i)$ **and** $(GT_{ci}$ **hasCompatibleAction** $GI_i)$ **implies** $(GI_{ci}$ **hasCompatibleAction** $GI_i)$

This relation of Goal Templates, Goal Instances, and their action compatibility allows us to apply Action Knowledge Discovery on the Goal Template level as a pre-filter function for GG Discovery. Therefore, we define a action knowledge ontology, which is referred to as the "Cooperative Knowledge Ontology". This ontology defines a concept "cooperativeGoalGroup" with 2 attributes: set-valued "hasCooperativeGoal" wherein all Goal Templates with compatible actions are hold, and "hasCooperativeGoal Constraints" where WMO GG Mediators can be possibly occurring mismatches between Goal Templates in a cooperativeGoalGroup. Instances of this concept define groups of

Goal Templates with action compatibility – for example a Goal Template for a buyer and one for a seller see SWF use case.[13]

For realizing the Cooperation Knowledge Filter, an API allows to retrieve the cooperativeGoalGroup of for the Goal Template GT_i of the initiating Goal Instance GI_i. The connection between a Goal Instance and its Goal Template is defined in the 'instanceOf' description notion of the Goal Instance. [14]All existing Goals Instances GI_c that are instances of a GT_{ci} have action compatibility to GI_i; those GI_c that have status 'open' (i.e. to be solved) are provided as input to the GG Matcher for determining the object compatibility to GI_i.

4.2.3.2 GG Matcher

The result of the Cooption Knowledge Filter (i.e. all GI_c with status 'open') is taken by the GG Matcher as input. For each GI_c, the object compatibility is checked, resulting in a ggcooperation as defined in section 4.1.1; for each cooperativeGoalGroup one sub-ggcooperation is created that holds one GI_c for each GT_{ci} defined in the cooperativeGoalGroup. Each sub-ggcooperation represents one possible cooperation of partners with regard to action and object compatibility of the Goal Instances assigned to Freds as electronic representatives.

For object matchmaking in the GG Matcher, the matching notions of object discovery are applied (see section 3.2.2.1), realized in VAMPIRE as described in section 3.2.2.2 which is used via the Vampire Web Service (see section 4.1.2.3). We do not discuss the technical realization of the GG Discoverer in detail, but provide some examples for usage of the different matching notions for object knowledge discovery, and the need for action knowledge discovery as a pre-filter.

Within the SWF Use Case, there are two types of Goals (with related services), so-called 'Purchase Goals' and 'Query Goals'. The former is concerned with purchasing a piece of furniture (postcondition) with delivering the purchased furniture to the buyer's

[13] see [Stollberg, 2004], section 3.3. The Cooperative Knowledge Ontology differs from the general structure of "Action Knowledge Ontologies" as defined in section 3.3.1 due to technical reasons (the API used for filtering is built for the structure of this ontology) However, the Cooperative Knowledge Ontology defines action compatibility for Goal Templates and can be understood as a specific type of action knowledge ontology.

[14] In the prototype, this information is defined in the relation of the non-functional properties of a Goal Instance. In the prototype, Goal Instances are defined as WSMO Goals, thus they do not have the description notion of 'instanceOf' (see SWF Goal Management in section 2.2.2).

shipping address (effect), while the latter is about finding product information of all offers for specific furniture (postcondition, there is not effect).

For the Purchase Goals, we use the Intersection Match for matchmaking of postconditions and effects. With the postcondition of a Purchase Goal we want to express that there shall be one purchase of a specific piece of furniture, not all purchases that a potential partner may offer; for the purchase of a potential partner to be matching, the objects shall not be contradicting – so this is exactly what the Intersection Match notion defines. For effect matching, we want to express that we want one delivery of the purchased item, and the delivery method shall not be contradicting. By using the Intersection match here, also GI_c are discovered that fulfil the matchmaking requirements, but possibly also other effects might be defined in GI_c. To exclude this, further information would have to be provided in the Discovery Request (e.g. use Intersection Match and there shall be no other effects in GI_c).

For the Query Goals, we use the PlugIn match for postcondition matchmaking. The Goal Instance postcondition shall express that product information for all available products shall be provided by a potential partner – so precisely what the PlugIn Match notion allows to determine. There can be other objects in the postcondition of a matching GI_c, but these are not of interest. For matching of effects in Query Goals, we use the Exact Match: there shall be no effects, so the notion of effects in the initiating Goal Instance GI_i is modelled as “false”. In combination with the Exact Match (requires equivalence of request(X) and result(X)), only GI_c match that have “false” as the logical expression in their effects notion.

The need for the Cooperation Knowledge Filter that pre-selects potentially matching Goal Instances according to action compatibility can be verified by an example. We want to discovery a “Seller Goal Instance” for a “Buyer Goal Instance” – in order to establish a cooperation between a buyer and a seller for purchasing, which consists of the compatible actions ‘buy’ and ‘sell’. The following provides possible postcondition definitions of Buyer 1 (buy a red chair made of wood, height 90, with 4 legs), Buyer 2 (buy a chair made of wood with 4 legs) and a Private Seller (sell a chair, height 90, with 4 legs). The models refer to the domain ontologies and the knowledge base defined in the SWF use case [Stollberg, 2004].

```
// Buyer 1:
postcondition
 axiom bgi1Postcondition
     nonFunctionalProperties
          dc:description hasValue " buy a red chair made of wood, height 90, with 4 legs "
     endNonFunctionalProperties
     definedBy
```

```
          exists ?PCID(?PCID memberOf xsd:integer) and
          exists ?PurchaseItem(?PurchaseItem[
              item hasValue ?PurchaseFurniture
          ] memberOf swfmo:product) and
          exists ?PurchaseFurniture(?PurchaseFurniture[
              height hasValue 90,
              material hasValues {wood},
              color hasValue "red",
              numberOfLegs hasValue 4
          ] memberOf furn:chair) and
          ?X[
              purchaseContractID hasValue ?PCID,
              purchaseItem hasValue ?PurchaseItem,
              buyer hasValue kb:MichaelStollberg,
              purchasePayment hasValue kb:MSCreditCard
          ] memberOf swfmo:purchaseContract .


// Buyer 2:
postcondition
 axiom bgi2 Postcondition
     nonFunctionalProperties
          dc:description hasValue "buy chair made of wood with 4 legs "
     endNonFunctionalProperties
     definedBy
          exists ?PCID(?PCID memberOf xsd:integer) and
          exists ?PurchaseItem(?PurchaseItem[
              item hasValue ?PurchaseFurniture
          ] memberOf swfmo:product) and
          exists ?PurchaseFurniture(?PurchaseFurniture[
              material hasValues {wood},
              numberOfLegs hasValue 4
          ] memberOf furn:chair) and
          ?X[
              purchaseContractID hasValue ?PCID,
              purchaseItem hasValue ?PurchaseItem,
              buyer hasValue kb:IoanToma,
              purchasePayment hasValue kb:ITCreditCard
          ] memberOf swfmo:purchaseContract .



// Private Seller:
postcondition
 axiom sgi1Postcondition
     nonFunctionalProperties
          dc:description hasValue " sell a chair, height 90, with 4 legs "
     endNonFunctionalProperties
     definedBy
          exists ?PCID(?PCID memberOf xsd:integer) and
          exists ?PurchaseItem(?PurchaseItem[
              item hasValue ?PurchaseFurniture
          ] memberOf swfmo:product) and
          exists ?PurchaseFurniture(?PurchaseFurniture[
              height hasValue 90,
              numberOfLegs hasValue 4
          ] memberOf furn:chair) and
          ?X[
              purchaseContractID hasValue ?PCID,
              purchaseItem hasValue ?PurchaseItem,
              seller  hasValue kb:UweKeller,
              purchasePayment hasValue swfmo:paymentMethod
 ] memberOf swfmo:purchaseContract .
```

Listing 5: GG Matcher Example

Consider the Goal Instance of Buyer 1 being the initiating Goal Instance GI_i; as a matchmaking notion, we use the Intersection Match as the goals are 'Purchase Goals' as explained above. Then, the GG Matcher would find the Goal Instance of Buyer 2 and the one of the Private Seller as matching GI_c – as the objects defined in the logical expressions of both postconditions are not contradicting (its true if you take a closer look – you can also find other examples within the testing of the GG Discoverer in section 4.1 of the SWF use case document). So, we would find two buyers (Buyer 1 and Buyer 2) as potential cooperation partners – which is not what we want.

With the Cooperation Knowledge Filter, the Goal Template for Buyer 2 would not defined in a cooperativeGoalGroup with the Goal Template for Buyer 1 – both have the action 'buy' which is not compatible. When using the Cooperation Knowledge Filter as a pre-filter for the GG Matcher, only the Private Seller Goal Instance would be found as a potential cooperation partner because of compatibility with respect to action and object knowledge.

4.2.4 Interfaces

Regarding the technical integration of the discoverer modules into the overall SWF Architecture, we define the INPUT denoted by arrows labeled with 'input', the OUTPUT and the RESOURCES which are understood as information needed for the discovery functionality denoted by arrows labeled with 'use' (see Figure 5).

4.2.4.1 Input and Ouput

The input of the GG Discoverer is one Goal Instance, which is a newly created Goal Instance (status = open). The diction of such Goal Instances is done by a Selection Engine that scans the system for new Goal Instances (permanently or event driven – based on the existing FRED Selection Engines). The GG Discoverer receives the complete WSML description of the Goal Instance, which is then parsed into the respective technical representations needed for the discovery mechanisms.

The output of the GG Discoverer is a set of Goal Instances. For each discovered Goal Instance, only the identifier is returned. This is then kept in the ggcooperation defined and managed in the SWF Discovery Manager (see section 4.1.1).

4.2.4.2 Resources

The GG Discoverer needs the following Resources:

> *Cooperative Goal Knowledge*: for the Cooperation Knowledge Filter, the Cooperative Knowledge Ontology has to be accessible. The ontology is defined as

Smart Object, and thus is accessible via the SMO API is the SWF system. Note that this functionality is only usable when the GG Discoverer is located within the SWF system.

1. *Ontologies:* for the object matchmaking in the GG Matcher, the domain ontology definitions are needed. As explained in section 3.2.2.2.1, the only required ontological knowledge needed for matchmaking with VAMPIRE is defined in the universe, optionally in the knowledge base. This is created a system setup, so that no import of ontologies is required.

   For terminological interoperability, the Goals have to use the same ontologies. The used ontologies are only specified in the Goal Template; no additional ontologies are used in the Goal Instance specification. The usage of the same ontologies in compatible Goal Templates is performed in the definition of a Cooperative Goal, so that no additional check is needed within the GG Discoverer.

2. *Mediators:* GG Discovery deals with 2 types of WSMO Mediators: OO Mediators used for terminology mismatch handling in the used ontologies, and GG Mediators for resolving mismatches between compatible Goals. Similar to ontologies, mediators are only specified in the Goal Templates. They have to be made accessible, so that the respective mediation services can be executed during cooperation execution. GG Mediators are defined in the Cooperative Goal Ontology with the "hasCoopertiveGoalConstraints"; the mediation service defined in a GG Mediator is called as a third party into a meeting, so that no additional efforts have to be undertaken for handling GG Mediators in the GG Discoverer.

   Mediators are only considered on a conceptual level, but not realized in SWF.

4.3 GS Discoverer

The second step in the resolution process is GS Discovery which detects suitable services for automated goal resolution separately for each cooperation partner identified in GG Discovery. Analogous to Web Service Discovery, GS Discovery returns a set of services that a partner can use for automated goal resolution and thus realizes the WSMO approach for Web Service Discovery [Keller et al., 2004].

4.3.1 Overview

The following defines the GS Discoverer as a specialization of the general WSMO discoverer architecture defined in section 3.1. Here, we explain realization of the GS Discoverer of the prototype: the final specification will be explained in subsequent documents.

4.3.1.1 Discovery Request

Assigned WSMO Resource: Goal Instance

> This are the individual Goal Instances defined in a ggcooperation as the result of GG Discovery. GS Discovery is executed for each Goal Instance separately, so that the discovery functionality of the GS Discoverer is independent of the SWF system.

Action Knowledge: Service Ontology

> defines action compatibility of Goal Templates and SWF Services

Object Knowledge: WSMO description of Goal Templates, Goal Instances, and WS Services

Matching Notion: depending on assigned Goal Instance

4.3.1.2 Discovery Result

set of Services

For the Discovery Request, a set of matching services is determined according to the information provided in the request. The GS Discovery Result is returned to the SWF Discovery Manager which creates the gscooperation as defined in section 4.1.1.2. Note that, due to the overall SWF Conceptual Model, the GS Discovery Result is a set of services that have equal actions to the GS Discovery Request, not compatible actions. This means that for a buyer goal GS Discovery detects buyer services, not seller services.

4.3.2 Functionality

Figure 16 shows the overview of the GG Discoverer, which is separated into 2 steps. The first step is the **Pre-Selector**; the Discovery Result is a set of possibly usable Services detected by matching Goal Templates (WSMO Goals) with all Services existing in the Service Repository. When retrieving the services from the Service Repository to be checked in the Pre-Selector, a filter is applied that pre-filters the services according to the action knowledge and usage permissions. This step is performed whenever a new Goal Template or Service is added to the system. In the second step the **GIS Matcher** detects which Services of the Pre-Selector Discovery Result match the Goal Instance, returning a reduced set of Services as the overall GS Discovery Result.

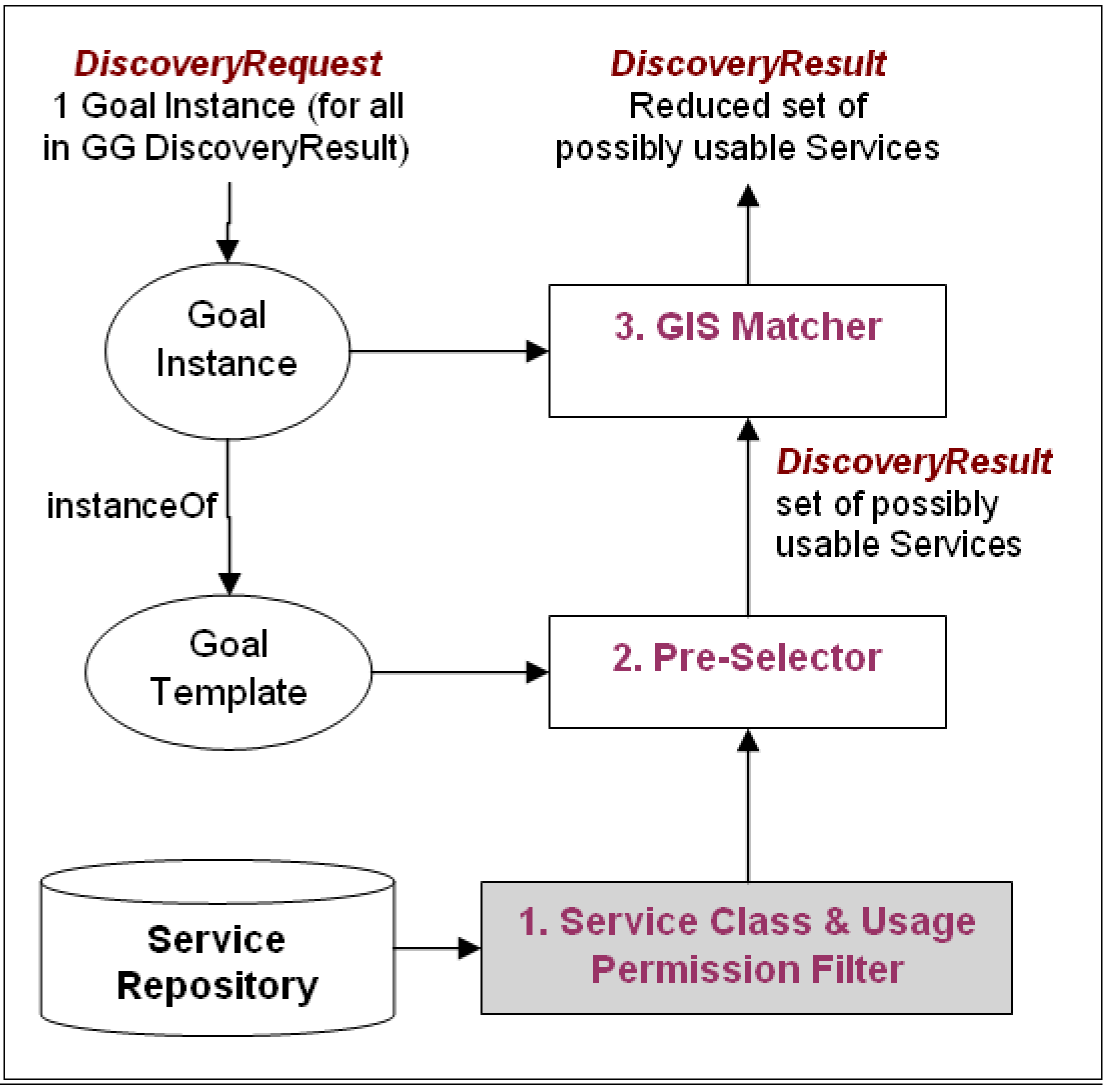

DiscoveryRequest
1 Goal Instance (for all in GG DiscoveryResult)
DiscoveryResult
Reduced set of possibly usable Services
Goal Instance
3. GIS Matcher
instanceOf
DiscoveryResult
set of possibly usable Services
Goal Template
2. Pre-Selector
Service Repository
1. Service Class & Usage Permission Filter

Figure 16: GS Discovery Overview

The reason for separating GS Discovery into sub-steps is first that the Pre-Selector provides an extended action knowledge pre-filter, and secondly it improves performance because not for all Goal Instances the whole Service Repository has to be searched, and thirdly SWF applies a WSMO-compliant Web Service Discovery technique within the Pre-Selector Matchmaker which can easily be replaced by other WSMO Discoverers.

The GS Discovery Result is a set possibly usable Services for one partner of a potential collaboration. The GS Discovery Result is tuples of GI with a set of Services that is returned to the SWF Discovery Manager which determines the final gscooperation of the separate GS Discoverer invocations.

4.3.3 Discovery Mechanisms Specification

The following explains the realization of the GS Discoverer. First, we provide a formal specification of the GS Discoverer in order to verify the architecture of the GS Discoverer with regard to the specification and relationships of the relevant SWF components and the expected discovery behavior.

4.3.3.1 Rationale of GS Discoverer Architecture

A SWF Goal Template is described as a WSMO Goal, i.e. with postcondition and effects as the main constituting description notions. Due to the relation between a Goal Template `GT` and a Goal Instances `GI` in SWF (logical dependency: `GI` **`instanceOf`** `GT`; technical connection via identifiers, see section 4.2.3.1), we can, first, determine the Goal Template GT_i for the Goal Instance GI_i for which the GS Discoverer is invoked, and, secondly, it is ensured that if a Service `WS` matches GI_i, then it matches GT_i as well:

$$(WS\ \mathbf{matches}\ GI_i)\ \mathbf{and}\ (GI_i\ \mathbf{instanceOf}\ GT_i)\ \mathbf{implies}\ (WS\ \mathbf{matches}\ GT_i)$$

This implication does not hold the other way around, as there might be a Service `WS` that matches the Goal Template GT_i, but not the Goal Instance GI_i because the restrictions in GI_i can be narrower than in GT_i. The following example clarifies this relation. The listings are snippets from Goal Templates, Goal Instances, and Service descriptions of the SWF Use Case [Stollberg, 2004]; the discussion of the matchmaking behavior is provided below.

```
// Web Service:
postcondition
 axiom bs1Postcondition
     nonFunctionalProperties
          dc:description hasValue " a purchase contract for a chair"
     endNonFunctionalProperties
     definedBy
```

```
          exists ?PurchaseItem(?PurchaseItem[
               item hasValue ?PurchaseFurniture
          ] memberOf swfmo:product) and
          exists ?PurchaseFurniture memberOf furn:chair and
          ?X[
               purchaseItem hasValue ?PurchaseItem,
               …,
          ] memberOf swfmo:purchaseContract .


// Goal Template:
postcondition
 axiom bgt1Postcondition
     nonFunctionalProperties
          dc:description hasValue " a purchase contract for a piece of furniture"
     endNonFunctionalProperties
     definedBy
          exists ?PurchaseItem(?PurchaseItem[
               item hasValue ?PurchaseFurniture
          ] memberOf swfmo:product) and
          exists ?PurchaseFurniture memberOf furn:furniture and
          ?X[
               purchaseItem hasValue ?PurchaseItem,
               …,
          ] memberOf swfmo:purchaseContract .


// Goal Instance 1:
postcondition
 axiom bgi1Postcondition
     nonFunctionalProperties
          dc:description hasValue " a purchase contract for a piece of chair"
     endNonFunctionalProperties
     definedBy
          exists ?PurchaseItem(?PurchaseItem[
               item hasValue ?PurchaseFurniture
          ] memberOf swfmo:product) and
          exists ?PurchaseFurniture memberOf furn:chair and
          ?X[
               purchaseItem hasValue ?PurchaseItem,
               …,
          ] memberOf swfmo:purchaseContract .


// Goal Instance 2:
postcondition
 axiom bgi2Postcondition
     nonFunctionalProperties
          dc:description hasValue " a purchase contract for a piece of table"
     endNonFunctionalProperties
     definedBy
          exists ?PurchaseItem(?PurchaseItem[
               item hasValue ?PurchaseFurniture
          ] memberOf swfmo:product) and
          exists ?PurchaseFurniture memberOf furn:table and
          ?X[
               purchaseItem hasValue ?PurchaseItem,
               …,
          ] memberOf swfmo:purchaseContract .
```

Listing 6: GS Discovery Matchmaking Behavior with Implication Match

Listing 6 shows a Service description for a purchase contract for a chair; a Goal Template description for a purchase contract for a furniture; a Goal Instance 1 for a purchase contract for a chair; and a Goal Instance 1 for a purchase contract for a table (the differences are marked by color in the listing). These resources relate to 'Purchase Goals' in the SWF use case as introduced in section 4.2.3, so the Intersection Match is used as the matchmaking notion is a Discovery Request. Therefore, the following matchmaking behavior holds:

- the `WS` matches `GT`, as there is no contraction in the objects (due to the generic instances defined in the universe, there is a purchase contract existing that has chair; this can satisfy the postcondition of `WS` as well as the one of `GT`).
- $\mathtt{GI}_1$ is a valid instance of `GT`, restricting the purchase item to chairs (a subconcept of furniture); the `WS` matches $\mathtt{GI}_1$
- $\mathtt{GI}_2$ is a valid instance of `GT`, restricting the purchase item to tables (a subconcept of furniture); the `WS` does not match $\mathtt{GI}_2$

The matching behavior of Goal Templates, Goal Instances, and Services for matching notions with universally quantified objects (e.g. plugin, subsumption, and exact match) is similar; in addition, the matchmaking is only successful if the request resource (i.e. the goal notion) is more specific then the result resource (the web service notion) – see section 3.2.2.1.2.
This matchmaking behavior that results from the definition of SWF resources and relation determines the architecture of the GS Discoverer in order to retrieve valid, high quality GS Discovery Results.

As the first step, the GS Discovery Pre-Selector provides a pre-selection of services that match the Goal Template $\mathtt{GT}_i$ of the initiating Goal Instance $\mathtt{GI}_i$. The Discovery Result of the Pre-Selector `DRes-GSPR` serves as input for the GIS Matcher which determines matching of the Goal Instance with the services in `DRes-GSPR`, resulting in the final GS Discovery Result `DRes-GS`. The Pre-Selector contains an extended action filter mechanism, and performs object matchmaking between Goal Templates and all Services available in the system at compile time (i.e. whenever a Goal Template or Services is added or removed from the repository). The latter is an expensive operation with reagrd to the matchmaking itself as well as to the amount of resources to be matched. The GIS Matcher performs object matchmaking at runtime; action knowledge discovery is not needed here as it is the same as in the corresponding Goal Templates, and the expenses for object matchmaking are decreased to a minimum, as `DRes-GSPR` provides the minimal set of services that have to be considered for matchmaking. [15]

4.3.3.2 Pre-Selector

The Pre-Selector is invoked whenever a new Goal Template or a Service is added to or removed from the system (notification via listeners, see 4.3.4.3). The Pre-Selector Discovery Result `DRes-GSPR` is kept in a internal database, and used as input for the GIS Matcher.

The GS Pre-Selector encompasses two different matchmaking technologies: the "Service Class & Usage Permission Filter" and the "Pre-Selector Matchmaker". The following explains the realization of both. According to the general design rule "most expensive operation at last", the Filter is located before the Matchmaker.

4.3.3.2.1 Service Class & Usage Permission Filter

This is a filter mechanism that pre-filters the services from the Service Repository according the action knowledge and the usage permission of the initiating Goal Instance GI_i.

Within the action knowledge filtering, only those service are being passed which have action equality with GI_i. The reason is that with GS Discovery in SWF, we want to retrieve those services that a partner might can you for automated cooperation execution; for instance, this means we want to discover a "buyer service" for a "buyer goal instance", not a seller service. This is ensured by action equality filtering, which is realized as a filter on basis of the action knowledge ontology.[16]

Secondly, the services are filtered on basis of the service usage permission that a Fred as the owner of the initiating Goal Instance GI_i has. The service usage permission is a property in the social description of a Fred that allows preventing erroneous service usage. For example, a Fred that is the electronic representative of a private seller in the marketplace described in the SWF use case has usage permission for the "private seller service", but not for the "IKEA service" wherefore only the IKEA-Fred has usage permission.

[15] The SWF prototype (see footnote 12) does only realize action knowledge filtering and object matchmaking between Goal Instances (Active WSMO Goals, see 2.2.2.2) and Services; however, these are the essential technologies, and realization of the full GS Discoverer is only extending these techniques.

[16] The SWF prototype therefore uses a 'Service Ontology' that defines a hierarchy of goals and services according to their actions (see section 3.5.1 in [Stollberg, 2004]). This ontology is a preliminary version of a overall action knowledge ontology of the SWF use case.

4.3.3.2.2 Pre-Selector Matchmaker

The Pre-Selector Matchmaker performs object matchmaking between the Goal Template `GT`$_i$ that the initiating Goal Instance `GI`$_i$ is an instance of, and the services from the Service Repository that have passed the Service Class & Usage Permission Filter.

For matchmaking, the matching notion defined in the GS Discovery Request is used. The object matchmaking works similar to the GG Matcher in GG Discovery, i.e. via the VAMPIRE Web Service using the Implementation of WSMO Discovery as explained in section 3.

The result of the Pre-Selector Matchmaker represents the GS Pre-Selector discovery result `DRes-GSPR` that is kept in an intermediary data base.

As SWF Goal Templates are structurally equivalent to WSMO Goals, and SWF Services are described as WSMO Web Services (according to WSMO 1.0 [Roman et al., 2004]), we consider the Pre-Selector Matchmaker as an implementation of WSMO Discovery for Goals and Web Services for WSMO v1.0; it determines a set of possibly usable web services by matching postconditions and effects, as already been described in [Keller et al., 2004b].

4.3.3.3 GIS Matcher

The GIS Matcher takes the GS Pre-Selector discovery result `DRes-GSPR` and performs object matchmaking with the initiating Goal Instance `GI`$_i$.

Apart from postcondition and effects that are instantiated from the corresponding Goal Template, a SWF Gaol Instance carries a submission as a additional description element. The submission holds those instance data that the Goal Instance owner is willing to submit as input to a service; these data are ontology instances according to the used domain ontologies. The conceptual reason for adding the notion of submission to SWF Goal Instances, which is contracting to WSMO v1.0 goal specifications, is to delegate a task complete to an agent by assigning a Goal Instance to it: input information for a service is needed anyway, and we do not want to force the user to provide input information for service execution if this is not explicitly needed.

The submission is used to test whether the capability descriptions that describe a service before it is executed, i.e. the preconditions and assumptions can be satisfied by the Goal Instance. As the submission is defined as instance definitions, this test is realized by checking whether the submission of `GI`$_i$ can satisfy the preconditions and assumptions of a service. Therefore, three scenarios can appear:

1. the submission exactly satisfies the precondition and assumptions of the service capability
2. the submission over-satisfies the precondition and assumptions of the service capability, meaning that the submission contains more information than requested by the precondition and assumption
3. the submission over-satisfies the precondition and assumptions of the service capability, meaning that the submission does not contain all information that are requested by the precondition and assumption

While in the first 2 cases, no additional user interaction is required, for the third case the user (who is the owner of the Fred that owns `GI`$_i$) can choose to either provide the additionally requested information, or to cancel the service usage when he does not or can want provide the additionally requested information. We understand this check as a proposal for handling the idea of adding the notion of input to WSMO Goals, in relation to the discovery framework defined in [Kifer et al., 2004].

If the submission test is successful, the object matchmaking technique of the Pre-Selector Matchmaker is applied again on the postconditions and effects of `GI`$_i$. This is necessary because of the GS Discovery matchmaking behavior discussed above. Note that the matchmaking notion used here is that same as used in the Pre-Selector Matchmaker for the corresponding Goal Template `GT`$_i$; this information is kept in the he GS Pre-Selector discovery result `DRes-GSPR`, so no additional information are required for the GIS Matcher object matchmaking.

4.3.4 Interfaces

According to the aspects explained for the GG Discovery, the following specifies the input, output, and the needed resources for the GS Discoverer. Note that the GS Discoverer works independently from the SWF system, and thus can be used within any other application.

4.3.4.1 Input and Output

As input, two types of resources are needed:

1) One Goal Instance. GS Discovery is invoked for all `GI` of a ggcooperation separately. The complete WSML description of the Goal Instance is taken as input.
2) All Services in the Service Repository. The Service Class and Usage Filter only receives the identifiers for filtering, while the Pre-Selector requires the complete WSML description of the Service for matchmaking.

4.3.4.2 Resources

The GS Discoverer needs the following Resources:

4. *Ontologies:* for the matchmaking, the domain ontology knowledge is needed. With realization of object matchmaking in VAMPIRE, only the universe definition and (optionally) the knowledge base are needed. These are created at system setup time, so that no further import of ontologies is needed in the GS Discoverer.

   For terminological interoperability, the Goals and Services have to use the same ontologies (used ontologies are only specified in Goal Templates; no additional ontologies are used in Goal Instance specification). This setting allows locating a filter mechanism that checks the usage of same ontologies in the Service Class and Usage Permission Filter; the check is achieved by comparing the values of the 'importedOntologies' description notion in Goal Template and Service descriptions.

5. *Mediators:* GS Discovery deals with 2 types of WSMO Mediators: OO Mediators used for terminology mismatch handling in the used ontologies, and WG Mediators for resolving mismatches between Goal Templates and Services as a pre-defined match.

   As ontologies, Mediators are only specified in the Goal Templates, while no further mediators are used within Goal Instances. The determination of using the same OO Mediators is incorporated in the 'same ontology usage filter' described above. A WG Mediator can be defined between a Service and a Goal Template as a pre-defined matching support if the Service and Goal Template do not match a priori. For applying a WG Mediator, it is invoked as a 3rd party in a meeting.

   Mediators are only considered on a conceptual level, but not realized in SWF.

4.3.4.3 Needed "Listeners"

For the Pre-Selector, "Listeners" inform the GS Discovery when a new resource for Pre-Selection is added to the system. In particular, a listeners a needed for newly added Goal Templates, and for for newly added Services.

These Listeners are incorporated into the deployment script for Goal Templates and Services (see section 2.2.4); whenever a new resource of this type is added, the Pre-Selector in the GS Discoverer is invoked.

4.4 WW Discoverer

For automated interaction of services as the realization of cooperative goal resolution, the Choreography Interfaces of cooperation partners have to be compatible with regard to behavioral models and messaging sequences. This is checked in WW Discovery as the last step in cooperation establishment determines, resulting in a global interaction model of the partners' services that allows automated execution of the cooperation.

4.4.1 Overview

The following defines the WW Discoverer as a specialization of the general WSMO discoverer architecture defined in section 3.1.

4.4.1.1 Discovery Request

Assigned WSMO Resource: set of services

> This is a single combination of services from a gscooperation wherefore the WW Discoverer determines choreography compatibility. WW Discovery is invoked for every possible combination of Services in the gscooperation, i.e. for the Services that have been detected as possibly usable for each cooperation partner by GS Discovery. These combinations are determined and managed by the SWF Discovery Manager (see section 4.1.1.3).

Action Knowledge: Choreography Compatibility Ontology

> this ontology defines compatibility patterns for service choreography descriptions (to be specified).

Object Knowledge: WSMO Choreography descriptions of the Services

Matching Notion: depending on Services and specification of choreography compatibility (not specified yet).

4.4.1.2 Discovery Result

boolean

The WW Discoverer has a boolean Discovery Result: TRUE if the input services are compatible with regard to their choreographies, FALSE otherwise. If the result is TRUE, then the Services are kept in wwcooperation; if the result is FALSE, the services are deleted from the list of services attached to the respective Goal Instance. This allows that in a meeting several services can be used – in case one service fails, it is not necessary to recall the whole meeting.

After execution of WW Discovery, the SWF Discovery Manager generates a Cooperation Contract that holds all information needed for a meeting for executing the cooperative goal resolution (see section 4.1).

4.4.2 Functionality

The functionality of WW Discovery is not elaborated so far – due to the not-existence of the WSMO Choreography Definition. However, the idea is to **test whether the Choreography Interfaces of 2 Services are compatible**, which means they have to be inverse – there is already ideas and related work to determine compatibility of process definitions and Abstract State Machines. Besides, as additional aspects or as a starting point, heuristic test on different aspects can be performed (usage of same / interoperable ontologies in the services, etc.).

The approach of testing the compatibility of Choreography Interfaces is limited for cooperation of 2 partners only. If we know that 2 Services have to talk to each other, each one is simply the Client who consumes the Choreography Interface of the other one. If there are cooperations of more than 2 partners, this approach does not work any longer – what we would need is a definition of a multi-party collaboration protocol that defines when which partner service is talking to which other service, etc. This is out of the scope of the SWF project, but the approach can be seen as a initial solutions towards determination of valid global interaction protocols for collaboration of multiple Web Services, which is the final aim of WSMO.

4.4.3 Specification

To be done ☺

4.4.4 Interfaces

As for the other SWF Discoverers, we specify the input, output, and needed resources for integrating the WW Discoverer into the overall SWF system.

4.4.4.1 Input and Output

The Input for the WW Discoverer are the choreography descriptions of the Web Services. The complete choreography description is required.

As the output is Boolean, there is no resources assigned to the output.

4.4.4.2 Resources

As the specification of the WW Discoverer mechanisms is not existing at this point of time, the required resources can not be specified. In principle, for WW Matchmaking, the domain ontology schemas and the 'Choreography Compatibility Ontology' are needed.

5 Conclusions

This document provides a specification of the technical realization of Semantic Web Fred with special focus on the SWF Discoverers as the core, WSMO-enabled components of the system.

We have explained those aspects of the overall SWF Architecture relevant for integration of the SWF Discoverers into the system. For realization of the discoverers, we have outlined an extended framework for WSMO Discovery and the technical realization in SWF. Then, we specified the architecture and discovery mechanisms for the distinct SWF Discoverers in detail – apart from WW Discovery which will be provided in a future version of SWF.

References

WSMX, The Web Service Execution Environment; WSMX Working Group homepage at: www.wsmx.org

Stollberg, M.; Herzog, R., Zugmann, P.: *SWF Framework*. Semantic Web Fred Deliverable D1, April 2004. available at: http://swf.deri.at/papers/SWF-D1-SWFFramework-final.pdf

Stollberg (ed).: *SWF Use Case*, most recent version available at: http://www.deri.at/research/projects/swf/usecase/

Roman. D., Lausen, H.; Keller, U. (eds.): *Web Service Modeling Ontology WSMO (Standard), final version 1.0*. WSMO Deliverable D2, WSMO Working Draft 20 September 2004; available at http://www.wsmo.org/2004/d2/v1.0/

Riazanov, A.; Voronkov, A.: *The design and implementation of VAMPIRE*. AI Communications 15(2), Special issue on CASC, pp. 91 -110, 2002.

Kifer, M.; Lara, R.; Polleres, A..; Zhao, C.; Fensel, D.; Keller, U.; Lausen, H.; Stollberg, M.: *A Logical Framework for Web Service Discovery*. Submitted to the ISWC 2004 Workshop on Semantic Web Services: Meeting the World of Business Applications.

Keller, U.; Stollberg, M.; Fensel, D.: *WOOGLE meets Semantic Web Fred*. Proceedings of the Workshop on WSMO Implementations (WIW 2004) Frankfurt, Germany, September 29-30, 2004. CEUR Workshop Proceedings, ISSN 1613-0073.

Keller, U.; Lara, R.; Polleres, A. (ed.): *WSMO Discovery.* WSML Working draft D5.1, 08 October 2004; available at: http://www.wsmo.org/2004/d5/d5.1/v0.1/20041008/

Herzog, R.; Zugmann, P.; Stollberg, M.; Roman, D.: *WSMO Registry.*, WSMO Working Draft D10, 26 April 2004.

Grimm, S.; Motik, B.; Priest, C.: Variance in e-Business Service Discovery. To be published, 2004.

de Bruijn, J. (ed): *The WSML Specification*, WMSL Working Draft 29 October 2004, avaliable at: http://www.wsmo.org/2004/d16/ .

Appendix – SWF Use Case Overview

Referring to the [SWF Use Case], this Appendix contains overviews in tabular format needed for development of the SWF Discoverers.

The abbreviations refer to the notions defined in the SWF Use Case, available at: http://www.deri.at/research/projects/swf/usecase/ .

A.1 Use Case Resources Overview

Table 4: Goal Templates Overview

Na me / ID	Postcondition	Effect	Compatible GT
BGT 1 *BuyFurnitureCredidCardGoal*	PC PF: furniture B: add, CC P: CC	Drop ship	SGT 1
BGT 2 *BuyFurnitureAnyPaymentGoal*	PC PF: furniture B: add, payment P: any	Drop ship or self collection	SGT 1, SGT 3
BGT 3 *QueryFurnitureRequestGoal*	Product Furniture: furniture Provider: any seller	-	SGT 2
BGT 4 *BuyFurniturePrivateGoal*	PC PF: furniture B: add, buyerpay-ment, S: Private Seller, Seller Payment P: any, fits Buyer-SellerPayment	Drop ship	SGT 3
SGT 1 *SellFurnitureGeneralGoal*	PC PF: furniture S: add P: any	Drop ship or self collection	BGT 1 , BGT 2
SGT 2 *QueryFurnitureProviderGoal*	Product Furniture: furniture Provider: any seller	-	BGT 3
SGT 3 *SellFurniturePrivateGoal*	PC PF: furniture S: Private, Add, P: any	-	BGT 4

Table 5: Goal Instances Overview

Na me / ID	Postcondition	Effect	Instance of GT
BGI 1 *111*	PC PF: f6 B: MichaelStollberg P: MSCreditCard	Drop ship	BGT1
BGI 1 *112*	PC PF: f19 B: MichaelStollberg P: MSCreditCard or MSCash	Drop ship or selfcollection	BGT2
BGI 3 *113*	Product F: storageFurniture, material = wood B: DieterFensel P: ITCash or ITCheck	-	BGT3
BGI 1 *114*	PC PF: f5 B: IoanToma P: ITCash or ITCheck	selfcollection	BGT4
SGI 1 (IKEA) *121*	PC PI: all IKEA products B: BuyerPayment S: IKEA P: all where BuyerPayment = IKEA.acceptsPaymentMethod	Drop ship or selfcollection	SGT1
SGI 1 (LEINER) *1212*	PC PI: all LEINER products B: BuyerPayment S: LEINER P: all where BuyerPayment = LEINER.acceptsPaymentMethod	Drop ship or selfcollection	SGT1
SGI 1 (KIKA) *1213*	PC PI: all KIKA products B: BuyerPayment S: KIKA P: all where BuyerPayment = KIKA.acceptsPaymentMethod	Drop ship or selfcollection	SGT1
SGI 3 *122*	Product F: furniture; S: any seller	-	SGT2
SGI 1 (Private) *123*	PC PI: UKProduct (f19) B: BuyerPayment S: UweKeller P: all where BuyerPayment = UweKel-ler.acceptsPaymentMethod	selfcollection	SGT3
SGI 1 (Private) *1232*	PC PI: MSBookShelf (f5) B: BuyerPayment S: MichaelStollbergSeller P: all where BuyerPayment = MichaelStoll-bergSeller.acceptsPaymentMethod	selfcollection	SGT3

Table 6: Services Overview

Na me / ID	Postcondition	Effect	Service Class / Usage Permission
BS 1 *buyerservice1*	PC PI: PF, Seller PF: furniture B: btA, stA, BuyerPayment S: SellerPayment P: any where BP = SP	Drop ship or selfcollection	Class: Buyer UP: all buyer
BS 2 *Buyerservice2*	Product F: furniture Provider: seller	-	Class: Buyer UP: all buyer
BS 3 *Buyerservice3*	PC PI: PF, PrivateSeller PF: furniture B: BuyerPayment S: SellerPayment P: any where BP = SP	selfcollection	Class: Buyer UP: all buyer
SS IKEA *sellerservice IKEA*	PC PI: all IKEA products B: BuyerPayment S: IKEA P: any where BP = IKEA-P	Drop ship (IKEADeliveray-Service) or selfcollection	Class: Seller UP: IKEA
SS LEINER *sellerservice LEINER*	PC PI: all LEINER products B: BuyerPayment S: LEINER P: any where BP=LEINER-P	Drop ship (LeinerDeliv-erayService) or selfcollection	Class: Seller UP: LEINER
SS KIKA *sellerservice KIKA*	PC PI: all KIKA products B: BuyerPayment S: KIKA P: any where BP = KIKA-P	Drop ship (Ger-manParcel) or selfcollection	Class: Seller UP: KIKA
SS 1 *sellerservice1*	PC PI: PF, any Seller PF: furniture B: BuyerPayment S: Add, SellerPayment P: any where BP = SP	Drop ship or selfcollection	Class: Seller UP: all seller
SS 2 *sellerservice2*	Product F: furniture, Provider: any Seller	-	Class: Seller UP: all seller (?)
SS 3 *sellerservice3*	PC PI: PF, any Seller PF: furniture B: BuyerPayment S: PrivateSeller, SellerPayment P: any where BP = SP	selfcollection	Class: Seller UP: privateSeller only

A.2 GG Discovery Matching Overview

The following lists the resources that should match

Table 7: GG Discovery on Goal Templates (Cooperative Knowledge)

	BGT1	BGT2	BGT3	BGT4	SGT1	SGT2	SGT3
BGT1					X		
BGT2					X		X
BGT3						X	
BGT4							X
SGT1	X	X					
SGT2			X				
SGT3		X		X			

Table 8: GG Matcher – Matching Goal Instances

	111	112	113	114	121	1212	1213	122	123	1232
111					X	X	X			
112					X	X	X		X	
113								X		
114										X
121	X	X								
1212	X	X								
1213	X	X								
122			X							
123		X								
1232				X						

A.3 GS Discovery Matching Overview

The following lists the resources that should match

Table 9: GS Pre-Selector (Goal Templates <-> Services)

	BGT1	BGT2	BGT3	BGT4	SGT1	SGT2	SGT3
Buyerservice1	X	X		X			
Buyerservice2			X				
Buyerservice3		X		X			
Sellerservice1					X		
Sellerservice2						X	
Sellerservice3							X
Sellerservice IKEA					X		
Sellerservice LEINER					X		
Sellerservice KIKA					X		

Table 10: GIS Matcher (Goal Instances <-> Pre-Selector Result)

	111	112	113	114	121	1212	1213	122	123	1232
Buyerservice1	X	X								
Buyerservice2			X							
Buyerservice3		X		X						
Sellerservice1					X	X	X		X	X
Sellerservice2								X		
Sellerservice3									X	X
Sellerservice IKEA					X					
Sellerservice LEINER						X				
Sellerservice KIKA							X			

A.4 WW Discovery Matching Overview

The following lists the resources that should match

Table 11: WW Discoverer Matchmaking (service compatibility)

	BS1	BS2	BS3	SS1	SS2	SS3	IKEA	LEINER	KIKA
BS1				X		X	X	X	X
BS2					X				
BS3				X		X			
SS1	X		X						
SS2		X							
SS3	X		X						
IKEA	X								
LEINER	X								
KIKA	X								