



## D.7.1: Experimentation and Evaluation Plan

Project Number	035200
Project Acronym	ARGUGRID
Project Name	ARGUmentation as a foundation for the semantic GRID
Instrument	STREP
Thematic Priority	Advanced Grid Technologies, Systems and Services
Start date	1 June 2006
Duration	36 months
Document Type (MD/TR/D/P/CD) <sup>1</sup>	D: deliverable
Document Distribution (PU/PP/RE/CO) <sup>2</sup>	PU: Public
Document Code <sup>3</sup>	ARGUGRID-GMV-PL-001
Version	v.2.13
Editor (Partner)	José A. Barba (GMV)
Workpackage(s)	WP7
Date of preparation	7 July 2009
Due date (if applicable)	30 November 2008
Date of delivery to the EC (if applicable)	13 December 2008
Number of pages	70
Internal Reviewers	Francesca Toni (Imperial), Thanassis Stournaras (cosmoONE)

<sup>1</sup> MD=management document; TR=technical report; D=deliverable; P=published paper; CD=communication/ dissemination.

<sup>2</sup> PU= Public; PP= Restricted to other programme participants (including the Commission Services); RE= Restricted to a group specified by the consortium (including the Commission Services); CO= Confidential, only for members of the consortium (including the Commission Services).

<sup>3</sup> This code is constructed as described on page 13 of the Project Handbook.

---

## D.7.1: Experimentation and Evaluation Plan

**José A. Barba (Editor)**

GMV AEROSPACE AND DEFENCE, S.A.

Isaac Newton, 11 P.T.M. Tres Cantos, 28760 Madrid, Spain

Email: [jabarba@gmv.com](mailto:jabarba@gmv.com) ( : +34 91 807 2100

---

### ABSTRACT

This document describes the methodology, procedures and environment that will be applied to test the ARGUGRID system.

A set of Test Cases has been created for validating and evaluating the expected functionalities, following two different but complementary approaches:

- A classical bottom-up approach that tests individual components, components integration and the final ARGUGRID platform.
- A qualitative approach that divides the Test Cases in “factory” tests for the basic functionalities, and “magazine” test for the functionalities that show where the ARGUGRID solution provides added value.

Evaluation will be focused only on “magazine” tests, and for this evaluation specific criteria have been defined, taking into consideration the identified potential markets.

Concrete examples of the scenarios defined in D.1.2 are explained and detailed in this document, together with the different actors that participate in the Test Cases, and the environment used for the validation and evaluation.

---

### REVISION HISTORY

Date	Version	Author (Partner)	Modification
21 March 2008	v 0.1	José A. Barba (GMV)	Creation and TOC draft.
11 April 2008	v 0.2	José A. Barba (GMV)	TOC modifications according to comments raised in London meeting (April 2008).
23 April 2008	v 1.0	José A. Barba (GMV)	TOC modifications. Request for inputs.
19 May 2008	v 1.1	José A. Barba (GMV)	Added contributions from: F.Toni, D.Gaertner (Imperial), T.Stournaras (cosmoONE), M.Morge (UNIFI)
09 Jun 2008	v 1.2	José A. Barba (GMV)	Added contributions from: F.Toni, D.Gaertner (Imperial), T.Stournaras (cosmoONE), J.Jarred (RHUL), M.Morge (UNIFI), M.Grammatikou (ICCS), V.Curcin(InforSense)
23 June 2008	v 1.3	José A. Barba (GMV)	Included comments from internal reviewers: F.Toni (Imperial), T.Stournaras

			(cosmoONE)
26 June 2008	v 1.4	José A. Barba (GMV)	Included comments from internal reviewers: F.Toni, D. Gaertner (Imperial)
27 June 2008	v 1.5	José A. Barba (GMV)	Included comments from internal reviewers: D. Gaertner (Imperial)
29 Sept. 2008	v 1.6	José A. Barba (GMV)	Addressing reviewers' comments after review 2: added table+ARGUGRID specific criteria+some new tests (with input from Maxime Morge (UNIFI), Francesca Toni (Imperial), Thanassis Stournaras (cosmoONE), Kostas Stathis (RHUL), Phan Minh Dung (AIT))
1 October 2008	v 1.7	Francesca Toni (Imperial)	Some revisions (section 3.3: completely new; section 4: GOLEM tests, CaSAPI tests): with input from Maxime Morge (UNIFI) Kostas Stathis (RHUL), Mary Gramatikou (ICCS) & Moustafa Ghanem (InforSense)and taking into account comments by all partners
6 October 2008	v.1.8	Francesca Toni (Imperial)	Some revisions (section 3.3); some changes to the CaSAPI tests; revised platform tests (section 4.3)
10 October 2008	v.1.9	Jose A. Barba (GMV)	Contributions integrated. Stable version for reviewers.
13 October 2008	v.1.10	José A. Barba (GMV)	Comments by Dorian Gaertner (Imperial) and Thanassis Stournaras (cosmoOne)
3 Nov. 2008	v.2.0	Francesca Toni (Imperial)	Restructured and expanded, following reviewers' and PO's comments on version v.1.10 and MB conference call on 3 November 2008
4 Nov. 2008	v.2.1	José A. Barba (GMV)	Some information added. Sections rearranged.
5 Nov. 2008	v.2.2	José A. Barba (GMV)	More contributions added. Mary Gramatikou (ICCS), Francesca Toni (Imperial), Stefano Bromuri (RHUL). Sections rearranged.
7 Nov. 2008	v.2.3	José A. Barba (GMV)	Some contents have been extended. Fields in Test Case description fixed to match the existing actors, criteria, etc... Contributions added: Maxime (UNIFI)
11 Nov. 2008	v.2.4	José A. Barba (GMV)	Changes in list of Test Cases. Contributions: Thanassis Stournaras (cosmoONE)
12 Nov. 2008	v.2.5	José A. Barba (GMV)	Minor UNIFI contributions added
12 Nov. 2008	v.2.6	Maxime Morge (UNIFI)	Minor UNIFI contributions added
18 Nov 2008	v.2.7	Francesca Toni (Imperial)	Made several comments, modified sections 2, 3, 4 modified description of CaSAPI tests and MARGO tests, incorporated market

			analysis (input from InforSense)
20 Nov 2008	v.2.8	José A. Barba (GMV)	Revision of contributions. Test names modified accordingly with type: “factory” or “magazine”. Fixed references and tables formatting.
2 Dec 2008	v.2.9	José A. Barba (GMV)	Added contributions from Maxime Morge (UNIPi), Francesca Toni (Imperial), Mary Grammatikou (ICCS), Nabeel Azam (InforSense).
10 Dec 2008	v.2.10	José A. Barba (GMV)	Final version to review. Added contributions from Stefano Bromuri (RHUL), Stella Kafetzoglou (ICCS).
18 Dec 2008	v.2.11	José A. Barba (GMV)	Added comments/revision from Francesca Toni, Dorian Gaertner (Imperial), Maxime Morge (UNIPi)
5 Jan 2009	v.2.12	José A. Barba (GMV)	More comments from Dorian Gaertner (Imperial). Final version.
7 July 2009	v.2.13	José A. Barba (GMV)	Step numbers in Magazine Tests fixed. Changed to PU (Public).

---

## Subject Requirement Specification

Contribution of this document to project aims and objectives	O12 & O13
Target audience	European Commission, Reviewers, ARGUGRID Consortium
Intellectual Property Rights (IPR) information	Full consortium: <ul style="list-style-type: none"><li>• Each partner is the sole owner of its own tests.</li><li>• Shared tests have shared ownership.</li></ul>

# TABLE OF CONTENTS

<b>EXECUTIVE SUMMARY .....</b>	<b>9</b>
<b>1 INTRODUCTION.....</b>	<b>10</b>
<b>2 EVALUATION AND VALIDATION METHODOLOGY .....</b>	<b>12</b>
2.1 Approach .....	12
2.1.1 Bottom-up approach .....	12
2.1.2 “Factory” – “Magazine” approach.....	12
2.2 Validation criteria for “factory” tests .....	13
2.3 Evaluation criteria for “magazine” tests.....	13
2.3.1 General Criteria.....	13
2.3.2 ARGUGRID-specific criteria.....	14
2.3.2.1 ARGUGRID aims.....	14
2.3.2.2 Evaluation Criteria .....	15
2.3.2.3 Market analysis .....	16
2.3.2.4 Measurement of evaluation criteria.....	18
2.3.3 Evaluation criteria and the ARGUGRID platform .....	19
<b>3 TESTING STRATEGY AND TEST CASES.....</b>	<b>20</b>
3.1 Test Case Identification.....	20
3.2 Test Case Description .....	20
3.3 Validation and Evaluation Reports.....	21
3.4 Suspension Criteria And Resumption Requirements .....	21
3.5 Testing Team. ....	22
<b>4 EVALUATION AND VALIDATION ENVIRONMENT .....</b>	<b>24</b>
4.1 ARGUGRID Scenarios.....	24
4.1.1 A brief description of the “RFP case” (“eProcRFP”).....	24
4.1.2 A brief description of the “Optimising eAuction parameters case” (“eProcOpt”).....	26
4.1.3 A brief description of the “Service Selection case” (“EOSelect”).....	27
4.1.4 A brief description of the “Service Orchestration case” (“EOComp”) .....	28
4.1.5 A brief description of the “Computer Assembly Investment case” (“Invest”) .....	28
4.2 ARGUGRID Integration Platform.....	28
4.2.1 EO Server.....	28
4.2.2 GRIA nodes .....	28
4.2.3 GOLEM containers .....	29
4.2.4 KDE server .....	29
4.3 Actors.....	29

<b>4.4</b>	<b>Web services .....</b>	<b>30</b>
4.4.1	Image providers.....	30
4.4.2	Image processing services.....	30
<b>4.5</b>	<b>Other components.....</b>	<b>32</b>
<b>5</b>	<b>TEST CASES .....</b>	<b>33</b>
<b>5.1</b>	<b>Classification of Test Cases .....</b>	<b>33</b>
5.1.1	“Magazine” tests .....	33
5.1.2	“Factory” tests.....	34
<b>5.2</b>	<b>“Magazine” tests: ARGUGRID Evaluation .....</b>	<b>35</b>
5.2.1	CaSAPI component .....	35
	Test Case MT.CaSAPI.SelectDominantSuppliers .....	35
	Test Case MT.CaSAPI.RankDominantSuppliers .....	36
5.2.2	MARGO component.....	37
	Test Case MT.MARGO.DecideJustifyEAuction.....	37
	Test Case MT.MARGO.SelectJustifyService .....	38
	Test Case MT.MARGO.ServiceComposition .....	38
5.2.3	Component integration .....	39
	Test Case MT.PLATON-GOLEM.Scalability .....	39
5.2.4	EO SCENARIO.....	40
	Test Case MT.EOSelect.ContractNegotiationMCP.....	40
	Test Case MT.EOSelect.WfContractNegotiationMCPR.....	41
	Test case MT.EOComp.ContractNegotiationMCP.....	43
	Test Case MT.EOComp.WfContractNegotiationMCPR.....	45
<b>5.3</b>	<b>“Factory” tests: Component Validation.....</b>	<b>47</b>
5.3.1	GRIA component .....	47
	Test Case FT_C.GRIA.ExecuteGRIAServices .....	47
	Test Case FT_C.GRIA.Registry.....	48
5.3.2	PLATON component.....	49
	Test Case FT_C.PLATON.Routing.....	49
	Test Case FT_C.PLATON.LoadBalancing.....	50
5.3.3	KDE component.....	50
	Test Case FT_C.KDE.Authoring.....	50
	Test Case FT_C.KDE.Argubroker.DPML_to_Prolog .....	51
	Test Case FT_C.KDE.Argubroker.WSMO_to_Prolog .....	52
	Test Case FT_C.KDE.Argubroker.WSML_to_Prolog .....	52
	Test Case FT_C.KDE.Argubroker.Prolog_to_DPML .....	53
	Test Case FT_C.KDE.Registry .....	53
	Test Case FT_C.KDE.Execution.1 .....	54
	Test Case FT_C.KDE.Execution.2.....	55
5.3.4	GOLEM component .....	56
	Test Case FT_C.GOLEM.AgentDeployment.....	56
	Test Case FT_C.GOLEM.ObjectDeployment.....	56
	Test Case FT_C.GOLEM.Interactions.....	57
<b>5.4</b>	<b>“Factory” tests: Integration Validation.....</b>	<b>57</b>

5.4.1	PLATON – GRIA integration .....	57
	Test Case FT_I.PLATON-GRIA.Discovery .....	57
5.4.2	PLATON – GOLEM integration .....	58
	Test Case FT_I.PLATON-GOLEM.DiscSuccess .....	58
	Test Case FT_I.PLATON-GOLEM.DiscFailure .....	58
5.4.3	GRIA – KDE integration .....	59
	Test Case FT_I.GRIA-KDE.Connectivity .....	59
	Test Case FT_I.GRIA-KDE.SLARetrieval .....	60
5.4.4	GOLEM – KDE integration .....	60
	Test Case FT_I.GOLEM-KDE.WorkflowCreation .....	61
	Test Case FT_I.GOLEM-KDE.ConcreteWorkflow .....	61
	Test Case FT_I.GOLEM-KDE.PartialWorkflow .....	62
	Test Case FT_I.GOLEM-KDE.Semantic.....	62
5.4.5	GOLEM – MARGO integration.....	63
	Test Case FT_I.GOLEM-MARGO.Service .....	63
<b>5.5</b>	<b>“Factory” tests: System Validation .....</b>	<b>63</b>
	Test Case FT_S.ARGUGRID.EndToEndConsumerScenario .....	63
<b>6</b>	<b>CONCLUSIONS.....</b>	<b>66</b>
	<b>ANNEX I: TEST CASE TEMPLATE.....</b>	<b>67</b>
	<b>ANNEX II: VALIDATION AND EVALUATION REPORT TEMPLATE .....</b>	<b>68</b>
	<b>ANNEX III: BUG REPORT TEMPLATE.....</b>	<b>69</b>
	<b>REFERENCES .....</b>	<b>70</b>

## **Executive summary**

The main objective of work package WP7 is to evaluate the developed ARGUGRID platform thoroughly with experiments that rely on scenarios developed in work package WP1.

The current document is the “Experimentation and Evaluation Plan”, identified as the D.7.1 deliverable of WP7 (“Experimentation and Evaluation”) led by GMV.

It describes the evaluation and validation plan for the platform developed across work packages WP2, WP3, WP5 and WP6. It also describes the methodology that will be used, the required testing environment and a full list of tests that will guide the validation of the system.

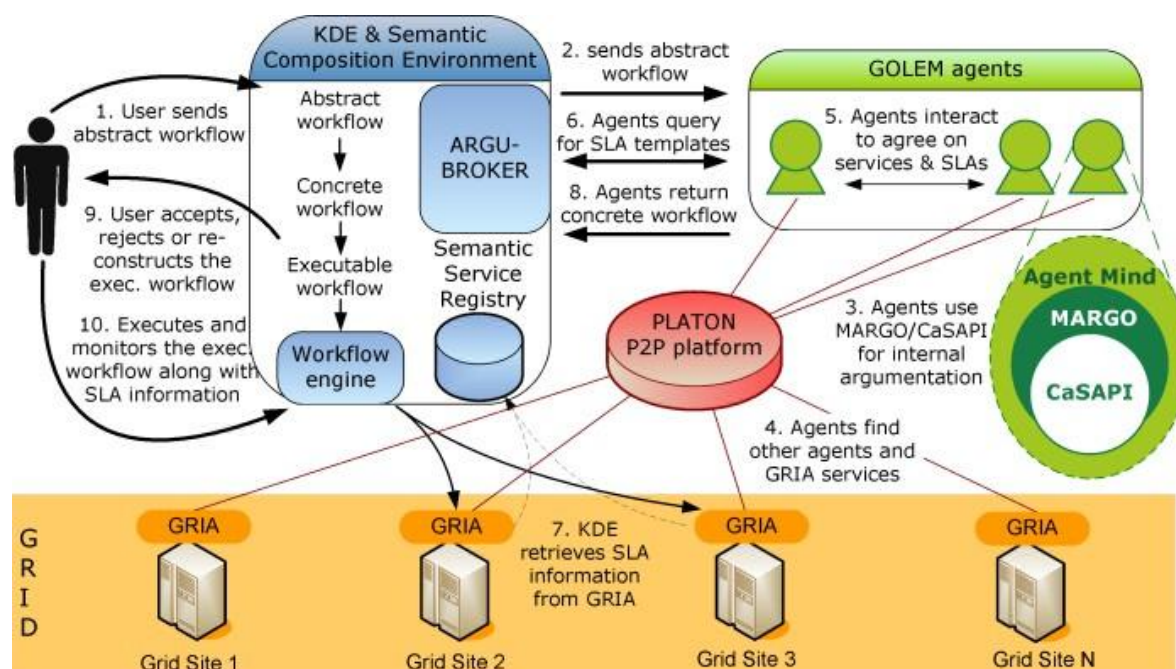
The methodology is engineered in such a way as to show the added value of the ARGUGRID approach with respect to the aims of the ARGUGRID project and in the context of the current “market” for Grid solutions, SOA architectures, argumentation methodologies and applications, Earth Observation and e-procurement products and applications.

# 1 Introduction

This deliverable describes the evaluation and validation plan for the ARGUGRID platform, consisting of the following components:

- The KDE workflow tools (by WP5, see deliverables D.5.2 and D.5.3)
- The MARGO and CaSAPI argumentation systems by WP2 (see deliverable D.2.2)
- The GOLEM agent platform by WP3 (see deliverable D.3.3)
- The Peer-to-Peer (P2P) PLATON platform by WP6 (see deliverables D.6.1 and D.6.2)
- The GRIA Grid platform, and its integration with PLATON to provide the ARGUGRID middleware, by WP6 (see deliverable D.6.2)

The interactions between these components, which comprise the full ARGUGRID platform, are outlined in the “Big Picture” deliverable and summarised in Figure 1 below.



**Figure 1: Global Picture of the ARGUGRID platform**

This deliverable describes the general testing methodology that will be used in the third year of the project to test components, integrations and the full ARGUGRID platform. It also specifies the required testing environment (in terms of the specific use-cases for the Earth-Observation and e-procurement ARGUGRID scenarios identified in WP1, see deliverable D.1.2). Finally, this document gives a full list of tests that will guide the correct validation of the system.

The methodology we propose to adopt is engineered in such a way as to show the added value of the ARGUGRID approach with respect to the aims of the ARGUGRID project and in the context of the current “market” for Grid solutions, SOA architectures, argumentation methodologies and applications, Earth Observation and e-procurement products and applications.

The tests are organised along two dimensions:

1. Whether they test components, integration of components, or the full ARGUGRID platform.
2. Whether they are “factory” tests (showing whether a specific component/integration or the full platform functions properly), or “magazine” tests (showing the added value of our solutions).

Factory tests can be seen as pre-requisites for magazine tests. The majority of the component and integration tests we propose are “factory” tests. All the platform tests are “magazine” tests.

The deliverable is organised as follows. In section 2 we describe our general evaluation and validation methodology. In section 3 we describe our testing strategy based on Test Cases. In section 4 we give details of the evaluation and validation environment. Section 5 details the full list of Test Cases that will be used for validation and evaluation. Finally, in section 6 we present our conclusions.

## 2 Evaluation and Validation Methodology

### 2.1 Approach

Our evaluation and validation methodology builds upon a number of tests, described in detail in section 5. These tests contain all the information required for validating the basic functionalities, and also for evaluating the main functionalities, considered as the added value of the ARGUGRID platform.

Our methodology integrates two dimensions: a bottom-up approach, and a distinction between “factory” and “magazine” tests.

#### 2.1.1 Bottom-up approach

According to with this widely used software testing approach, three different levels are defined:

- Component testing: each software component of the system is tested to verify its correct implementation.
- Integration testing: the interfaces and interaction between integrated components will be tested.
- System testing: to test the completely integrated system to verify the system's compliance with its specified requirements.

This strategy will allow testing the platform before it is completely finished.

#### 2.1.2 “Factory” – “Magazine” approach

In order to focus the evaluation on the functionalities that provide added value to the ARGUGRID platform, Test Cases are divided in two groups: “factory” and “magazine” tests:

- “Factory” tests aim at validating a specific component/integration or the full platform, showing correct functioning.
- “Magazine” tests aim at showing the added value of our solutions, in the context of ARGUGRID’s aims and the current “market”.

This approach (and the underlying distinction between factory and magazine tests) can be illustrated using the analogy of the evaluation of a car. The factory tests are the tests performed at the factory level, during the design and manufacturing process, in order to make sure that the all the car components and subsystems (engine, brakes, airbags, steering, dashboard instruments, etc.), and the car as a whole, work as desired. In a similar way, for the ARGUGRID evaluation, “factory” tests are designed to validate the components and their interaction, as well as the correct functioning of the overall system. The functionalities tested by means of factory tests may not show the added value of the ARGUGRID solutions. Instead, they can be considered just as prerequisites for the integration of the whole platform. Therefore, no evaluation will be done in these tests, just the validation of the described functionalities.

Magazine tests can also be illustrated using the car analogy. Some magazines publish often comparative evaluations of different cars. Sometimes they compare just two cars, other times they compare a family/category of cars. These evaluations are written by specialized journalists, who use their experience and knowledge to run a series of tests and actually drive the cars in question themselves to form a global view. These tests are designed to help them form an all-round opinion of the performance of the car. Almost all their evaluations are qualitative and usually along the lines of "This car has these pluses and these minuses". This is what we called "magazine" tests. Continuing this analogy, “magazine” tests have been designed to compare and evaluate the strongest points of the ARGUGRID platform against other solutions, in the identified markets. These tests will allow us to focus the evaluation on highlighting the added value of our solutions.

In the following sections, we identify validation criteria for “factory” tests and evaluation criteria for “magazine” tests.

## 2.2 Validation criteria for “factory” tests

A “factory” Test Case is deemed to be successful when the actual outputs, results or displays match the expected outputs, results or displays.

There are several kinds of possible validation criteria:

- GUI behaviour (GUI displays, print messages)
- File existence and consistency (file existence, file dates, file size, file content)
- Data base content (existence of information, metadata)
- Log file analysis (showing sequence of operations and intermediate results or status)
- Performance measurement (captured performance is not worse than expected)

For our “factory” tests, we will focus mostly on the second and third validation criteria above. Test Case results will be produced for the most significant validation tests performed.

## 2.3 Evaluation criteria for “magazine” tests

Besides the validation, an evaluation of the platform, its components and their integration will be performed, in the context of the given ARGUGRID scenarios, with focus on the EO scenario and, to some extent, the e-procurement scenario, and taking into account

- the added value of the ARGUGRID approach,
- with respect to the aims of the project, and
- in the context of the existing “market”.

Test Case results will be produced for the evaluation. While **validation criteria** measure if the system works in a normal way and produces the expected outputs, **evaluation criteria** allow measuring the suitability of the solutions and techniques underpinning the proposed system/platform. Suitability also involves comparison with other solutions and techniques within the current “market” for ARGUGRID (see section 2.3.2.3).

The evaluation criteria can be divided in “general criteria”, applicable to any software system (see section 2.3.1 below), and ARGUGRID-specific criteria (see section 2.3.2 below), referring to the specific added value of the ARGUGRID approach.

### 2.3.1 General Criteria

These are criteria that are commonly used to evaluate software, and in particular Grid solutions, and typically include:

- **Scalability**: indicates the ability of one system to be readily enlarged or to handle growing amounts of work in a graceful manner.
- **Performance**: measures the amount of useful work or outputs, compared to the time and resources used. Could be measured as a short response time or low utilization of resources.
- **Fault-tolerance**: is the ability of a system for continuing operating after the failure of some of its components.
- **Reliability**: is the ability of a component to perform its required functions under certain conditions. It is also the ability of such systems to manage failures in a controlled way, without negative consequences for the system. It can be reported in terms of a probability.
- **Manageability**: measures if a system can be managed or controlled in an easy way.

- **Interoperability:** is the ability of diverse systems or components to work together, to exchange data via a common set of exchange formats, and to use the same protocols.
- **Flexibility:** is the ability of a system to respond to internal or external changes. Uncertainty is a key element in the definition of flexibility.
- **Portability:** measures the capabilities of a system to be ported to different platforms in an easy way, e.g. onto other Grid platforms.
- **Implementation cost:** in our evaluation plan, refers to an estimation of the effort required to create one solution based on this technologies.

Although all these criteria are important for the ARGUGRID components, integrations and platform, as for any software, their fulfilment alone would not allow us to show the added value of the ARGUGRID solutions.

Within the third year of the project, of these criteria we will solely evaluate **Scalability**, as we consider this criterion as an important one for the ARGUGRID platform. This can refer to different aspects depending on the component/integration/full platform being evaluated. For example, in the PLATON P2P platform, scalability would refer to the number of peers, in GOLEM it would refer to the number of agents and containers, in MARGO to the size of the knowledge base held by the agent. We envisage that, in the third year of the project, we will focus on testing scalability of GOLEM, PLATON and their integration.

### 2.3.2 ARGUGRID-specific criteria

These criteria are meant to help assessing the validity of the specific approach taken by ARGUGRID and stress the benefits of the ARGUGRID approach with respect to the aims of ARGUGRID in the context of the “market”. Below, we first recap the aims of the ARGUGRID project, give our evaluation criteria for the fulfilment of the aims, and link these criteria to the market within which we are positioning ARGUGRID.

#### 2.3.2.1 ARGUGRID aims

The general **aims** of the ARGUGRID project are

1. to provide a new model and methodology for the specification, creation, operation and dissolution of VOs, formed by individuals or institutions participating in the Grid and controlling resources/services, and represented in ARGUGRID as agent societies, for the sharing of resources and services over the Grid, whereby resources and services are managed by rational argumentative agents communicating, negotiating and arguing with one another;
2. to provide a Grid-based model for these agents, capable of deciding rational plans of actions to achieve their goals while taking into account their preferences and the utilities they assign to situations, and evolving and using a notion of trust to “filter” their interactions with other entities (typically agents and VOs);
3. to design an architecture for the semantic Grid based on the use of communicating agents and built on top of standard Grid middleware and web service standards, to support the creation and evolution of VOs as in 1 and the operation of agents as in 2;
4. to develop a Grid-based platform to support the implementation of the models, and allowing to study the viability and usefulness of the proposed ARGUGRID formal approach;
5. to experiment with and evaluate the models, methodology, service-oriented architecture and platform in the context of concrete application scenarios for e-business;

using the following techniques:

- For 1: argumentation techniques (in the form of argumentation-based protocols) to support VO formation (where the outcome of VO formation is understood as consisting of a workflow and a contract) and VO operation in the case of disputes (on the execution of the workflow or the fulfillment of the contract).

- For 2: argumentation techniques and decision-theoretic techniques, to deal with utility and preferences optimization as well as trust.
- For 3: a web service architecture.
- For 4: peer-to-peer computing.
- For 5: the expertise of the industrial partners within the project.

### 2.3.2.2 Evaluation Criteria

The criteria we consider are summarised by Table 1:

ARGUGRID functionalities Evaluation criteria	Decision-making	Negotiation of Workflows	Negotiation of Contracts	Trust	Dispute Resolution
Quality of Results	QoR for DM	QoR for NW	QoR for NC	QoR for T	QoR for DR
Automation Level	AL for DM	AL for NW	AL for NC	AL for T	AL for DR
Complexity of Knowledge	CoK for DM	CoK for NW	CoK for NC	CoK for T	CoK for DR

**Table 1: the ARGUGRID-specific evaluation criteria**

The general criteria on the rows of this table are specialised in the context of the functionalities offered by ARGUGRID, resulting in concrete criteria such as QoR for DM: “Quality of Results for Decision-Making”, AL for NW: “Automation Level for Negotiation of Workflows”, etc. The columns of Table 1 link directly to aims 1-2, as follows:

- *(argumentation-based) Decision-making* and *(argumentation-based) trust* for aim 2,
- *(argumentation-based) Negotiation of workflows*, *(argumentation-based) Negotiation of contracts* and *(argumentation-based) Dispute Resolution* for aim 1.

We focus here on aims 1-2 as we believe that these refer to the core benefits of using argumentation and thus allow highlighting the real added value of the project.

The fulfilment of aims 3-4 is important and instrumental to the achievement of aims 1-2 in a Grid setting, but peripheral within ARGUGRID as it is shared by many other Grid platforms and applications. Aim 5 is ignored here as its achievement is the goal of this deliverable and more generally WP7 to which this deliverable contributes.

Since our models for dispute resolution and trust will be developed during the third year, while we will be conducting evaluation, we will focus mostly on the criteria corresponding to the first three columns. In detail, fulfilment of these criteria amounts to:

- **Quality of Results:** are the solutions given by ARGUGRID suitable/innovative/better than the solution produced by competitors of ARGUGRID within the current “market”? We will focus on:
  - **Decision-making:** is our argumentation-based approach suitable/innovative/an improvement with respect to decision-making as currently supported by existing solutions for automatic service selection/composition over the Grid?
  - **Negotiation:** are our protocols for negotiation of workflows and contracts suitable/innovative/an improvement with respect to automatic service selection/composition over the Grid?
- **Automation level:** this aspect refers to the level of autonomy that our solutions afford, and how much they free human users from any direct effort/involvement in using the

Grid/service-oriented architectures. The higher the automation level, the higher the independence of user supervision. We will focus on:

- **Decision-making:** how much time is required for agents to come-up with a solution (service instance, workflow instance, decision on trust for other agents/services), in comparison with the time taken by human users using existing platforms/applications?
- **Negotiation:** how much time is required for agents to come-up with a solution (agreed workflow, agreed contract), in comparison with the time taken by human users using existing platforms/applications?
- **Complexity of knowledge:** this aspect covers both the amount of knowledge human users need to input to the system and the clarity of system outputs. Specifically, we are interested in answering the following questions:
  1. Are users put in a position to achieve more from the system, while at the same time being freed from “menial” effort? In particular, we will focus on *instructing* the system with knowledge/requirements specified by users.
  2. Are users put in a position to understand the output of the system, transparently from the underlying technologies used and intuitively? In particular, we will focus on the level of *explanation* that the user gets from the system.

In the case of Decision Making and Negotiation, answering these questions amounts to:

- **Decision-making:** what is the amount of information that users need to provide to “their” agents so that these agents can go about identifying the right solution (service instance, workflow instance, decision on trust for other agents/services), in comparison with existing platforms/applications? How does the output of the system compare with that provided by existing platforms/applications?
- **Negotiation:** what is the amount of information that users need to provide “their” agents with so that these agents can negotiate a satisfying workflow/contract, in comparison with existing platforms/applications? How does the output of the system compare with that provided by existing platforms/applications?

In the third year of the project, we will focus on the selected criteria, highlighted in the following table:

ARGUGRID functionalities	Decision- making	Negotiation of workflows	Negotiation of contracts	Trust	Dispute Resolution
Quality of Results	QoR for DM	QoR for NW	QoR for NC	QoR for T	QoR for DR
Automation level	AL for DM	AL for NW	AL for NC	AL for T	AL for DR
Complexity of Knowledge	CoK for DM	CoK for NW	CoK for NC	CoK for T	CoK for DR

**Table 2: Highlighted ARGUGRID-specific evaluation criteria**

### 2.3.2.3 Market analysis

Overall, we will evaluate the criteria in comparison with what we believe to be the current “**market**”. This amounts to organisations and solutions fulfilling business requirements of increasing complexity, as follows:

1. Traditional Grid and service-oriented architectures (SOA), where business functionalities and business processes are packaged as interoperable services.
2. Business-to-business (B2B) collaborations, supported by the negotiation of services of appropriate quality (QoR) to be delivered according to agreed service-level agreements (SLAs).
3. Open market, a widely distributed and dynamically changing setting for businesses as well as non-expert users to provide and request services. This market needs to create opportunities for all, both individually and collectively, to benefit from the economic interactions afforded by the open market.

Our QoR criteria (for the chosen functionalities of the ARGUGRID approach) are needed in order to fulfill the requirements of the first and second segment of the market. Our AL criteria (again for the chosen functionalities) are needed to support the automation needs required by the third segment. The CoK criteria are needed to guarantee that non-expert users can participate and benefit from the open market.

We will consider examples of the first segment of the market (traditional Grid/SOA) such as GRIA and solutions by other EU-funded projects (e.g. NextGRID). Competitors of our industrial partners (InforSense, GMV and cosmoONE) will be our examples of the second segment of the market (B2B collaborations). For the third segment (open market) we will consider projects such as CONOISE-G and MBC (<http://www.marketbasedcontrol.com/>), as well as research by academic groups that can be seen as a step towards supporting open markets (this includes, for example, research on argumentation-based decision-making by Toulouse University in France).

Our market-based evaluation is summarized by the Table 3, below. There, the list of organisations/solutions on the rows is by no means exhaustive.

		QoR for DM	QoR for NW	QoR for NC	AL for DM	AL for NW	AL for NC	CoK for DM
	ARGUGRID							
Traditional GRID/SOA	GRIA							
	NextGRID							
Industrial partners competitors	Taverna							
	Deimos							
	Business Exchanges							
Open Markets	CONOISE-G							
	MBC							
	Toulouse university							

**Table 3: ARGUGRID-specific evaluation criteria applied to other market solutions**

Taverna is a direct InforSense’s competitor. By comparing the traditional tools or procedures that our industrial partner’s competitors still use, we will be able to show the benefits of using the ARGUGRID platform. Deimos (one of GMV’s competitors) will be used to compare the benefits that GMV can obtain in the EO market by using the ARGUGRID platform. And in a similar way, Business Exchanges S.A (one of cosmoONE competitors) will also be used to compare benefits that cosmoONE can obtain from ARGUGRID.

For the relevant pairs criterion/organisation we will assess fulfilment of the criterion by the organisation. Not all pairs will be of interest. For example, cosmoONE is not providing solution to the creation of workflows, so none of the pairs of the form “X for NW”/cosmoONE will be considered. Similarly, GMV is not concerned with negotiation of contracts.

The evaluation we are planning to perform is summarised in Table 4: here, a tick stands for an analysis we want to perform. We plan to replace each tick either by F (fulfilled) or U (unfulfilled) with appropriate justifications, including for the ARGUGRID row appropriate comparison with other ticked boxes in the same rows.

		QoR for DM	QoR for NW	QoR for NC	AL for DM	AL for NW	AL for NC	CoK for DM
	ARGUGRID	✓	✓	✓	✓	✓	✓	✓
Traditional GRID/SOA	GRIA				✓	✓	✓	✓
	NextGRID				✓	✓	✓	
Industrial partners competitors	Taverna		✓			✓	✓	✓
	Deimos	✓				✓		✓
	Business Exchanges	✓			✓			✓
Open Markets	CONOISE-G		✓	✓	✓	✓	✓	
	MBC		✓	✓	✓	✓	✓	✓
	Toulouse university	✓						✓

**Table 4: Planned Market-driven evaluation**

Note that, in our evaluation of existing market solutions, we are neglecting some other features/criteria that others focus on (e.g. see the CONTRACT project and its focus on automatically detecting whether contracts are violated): the criteria that we consider though link to our aims, as we discussed earlier.

### 2.3.2.4 Measurement of evaluation criteria

We will adopt a quantitative approach for assessing the fulfilment of criteria for ARGUGRID, adopting the following comparative metrics:

- **Quality of Results:** amongst the solutions (decisions/workflows/contracts) computed by ARGUGRID, how many are correct with respect to the solutions provided by the identified market segment? How many correct solutions are computed by ARGUGRID and not by other

approaches? Here the knowledge of our participating experts will be invaluable, as they will specify what “correctness” means.

- **Automation level:** how long does the ARGUGRID platform (or components thereof) take to generate a decision/workflow/contract? How many steps involving user interaction are required compared with other solutions? Again, the knowledge of our experts will be invaluable, as they will specify how long the existing systems they deploy currently require.
- **Complexity of knowledge:** how much information is required to instruct the ARGUGRID platform (or components thereof) in comparison with existing systems? How long does a user need to assess the appropriateness of a solution provided by the ARGUGRID platform (or components thereof) in comparison with existing systems? Here too the knowledge of our experts will be invaluable, as they will specify the format of input/output the existing systems they deploy currently require.
- **Scalability:** this will be measured in terms of number of elements (agents, services, containers, peers) we can accommodate.

### 2.3.3 Evaluation criteria and the ARGUGRID platform

We plan to execute tests making use of the platform components/integrations as well as the whole platform as summarised by the following Table 5.

		SPECIFIC CRITERIA						GENERAL CRITERIA	
		QoR for DM	QoR for NW	QoR for NC	AL for DM	AL for NW	AL for NC	CoK for DM	SCALABILITY
COMPONENTS	GRIA								
	PLATON								
	KDE								
	GOLEM								
	CaSAPI	✓			✓			✓	
	MARGO	✓			✓			✓	
INTEGRATIONS	PLATON-GRIA (ARGUGRID MIDDLEWARE)								
	PLATON-GOLEM								✓
	GRIA-KDE								
	GOLEM-KDE								
	GOLEM-MARGO								
SYSTEM	ARGUGRID PLATFORM	✓	✓	✓	✓	✓	✓	✓	

Table 5: Evaluation criteria versus components matrix

### 3 Testing strategy and Test Cases

Our testing strategy will be based on Test Cases describing functionalities of our system. A Test Case defines the interactions between external actors and the system (or component) under consideration to accomplish a goal.

Test Cases can show different results depending on the Actor's preferences or the environment conditions. In such a case, different specialized Test Cases have been created. These specialized Test Cases share a common prefix name, and are sequentially numbered. E.g.: FT\_C.KDE.Argubroker.1, FT\_C.KDE.Argubroker.2 and FT\_C.KDE.Argubroker.3 Test Cases.

The integration testing principles follow a black-box testing approach where the operator does not need to have knowledge of the code to perform the tests.

#### 3.1 Test Case Identification

The Test Cases will be identified using the following naming convention:

TT.FC.Func[.n]

where:

- TT (test type) could be one of the following values:
  - MT for “Magazine” tests, designed to evaluate the platform and some components.
  - FT\_C for “Factory” tests, designed to validate Components,
  - FT\_I for “Factory” tests, designed to validate components’ Integration,
  - FT\_S for “Factory” tests, designed to validate the whole System.
- FC is the name of the functional component (e.g. PLATON or KDE), components (e.g. GRIA-KDE) or system/scenarios (e.g. ARGUGRID, EOSelect) providing the functionality to be tested;
- Func is the name of the functionality (e.g. ExecuteGRIAServices)
- n, optional, is the Test Case sequential number within TT.FC.Func, in case several tests are required for testing a functionality with different actors or different conditions.

Some examples of Test Cases are:

- **MT.ARGUGRID.Instructing:** “Magazine” test for ARGUGRID system, testing functionality related to the instruction of the system.
- **FT\_C.KDE.Execution.1:** “Factory” test for KDE component. Testing functionality related to execution of a service through KDE. This is also the first test in a set that will check this functionality, as indicates the “.1” at the end of the code.
- **FT\_I.GRIA-KDE.Connectivity:** “Factory” test for connectivity functionality related to the integration of GRIA and KDE components.

#### 3.2 Test Case Description

Test Cases will be described using the following fields:

- Test Case ID,
- Summary,
- Actors,

- Preconditions,
- Triggers,
- Steps (including inputs and outputs/expected outcomes for each step),
- Postconditions,
- Evaluation criteria (only applicable for “magazine” tests),
- Notes,
- Author, date & version

A template for Test Cases can be found in Annex I. Concrete instances of this template are given in section 5.2 (“magazine” tests), 5.3 (“factory” tests for component), 5.4 (“factory” tests for integration), and 5.5 (“factory” tests for the system).

### 3.3 Validation and Evaluation Reports

The verification and validation results shall be recorded in **Validation and Evaluation Reports** (see template in Annex II) together with the software components version used. In particular, each discrepancy detected during the test shall be recorded in the test reports with a reference to the associated bug report (see Annex III).

### 3.4 Suspension Criteria And Resumption Requirements

If any failure is noted during the tests, it will be analysed in order to identify the faulty component(s). The failure would be reported in a “bug report” including all available information allowing solving the problem (see template in Annex III). The “bug report” would be allocated to the team responsible for the suspected faulty component(s). Based on the information provided in the “bug report”, the team should perform a root cause analysis of the failure and propose corrective actions and a delivery date accordingly.

In case the root cause analysis contradicts the initial team assessment and implicates other components, the bug report is re-allocated to the correct work group after agreement of all parties.

Depending on the failure severity and its potential impact on the other integration tests, a decision is made to rule on the continuation or adjustment of the test campaign. The overall planning is updated accordingly.

Once the failed components or failed interactions are corrected and tested, they are made available for integration testing again. But, before resuming the integration tests normally, regression testing may be required. This need is analysed on a case-by-case basis taking into account the realised corrections. In most cases, regression tests will consist in re-running some relevant component/interaction tests that have already passed previously.

The following diagram provides a simplified view of the order in which integration-testing tasks are performed, and of the update procedure in the event of a test failure:

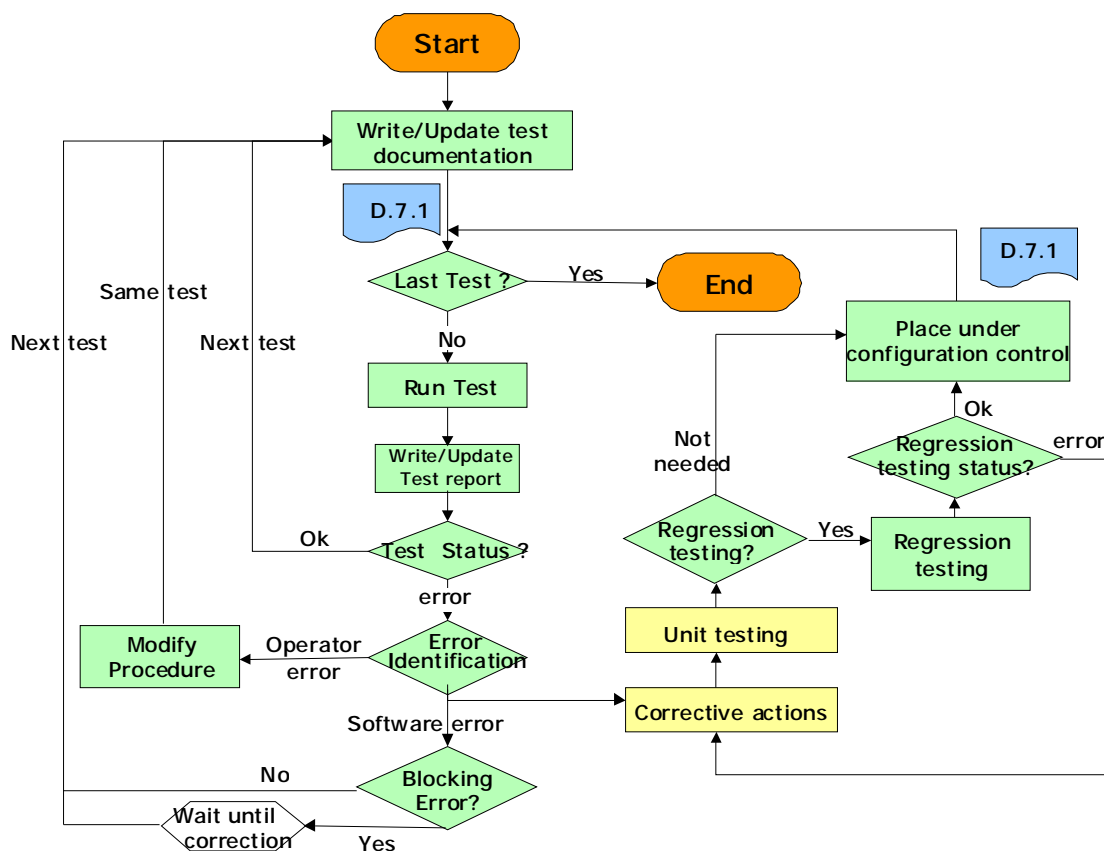


Figure 2: Integration process overview

This diagram includes alternative paths for common situations in validation procedures. These alternative paths are the following:

- Test passes (Diamond “Test Status”; “OK” case). In this case, next test has to be executed, until list of test is empty.
- Test fails (Diamond “Test Status”; “error” case). In this case, two different situations are possible:

- Operator error or procedure error (Diamond “Error Identification”, “Operator Error” case). In such a case, the test procedure has to be modified to avoid user’s error.

After modifying the procedure (and updating the Test document) a new iteration has to be done.

- Software error. In such a case, corrective actions have to be taken. After the error is identified and fixed, regression tests may have to be done to ensure that modifications do not introduce new errors in previously passed tests.

New versions of modified components have to be stored under configuration control, indicating all the information about the error and the Test Case that found it.

Finally, depending on whether the error blocks the execution of the following tests (Diamond “Blocking error”), the process has to wait until the error is fixed, or can continue with the list of tests.

### 3.5 Testing Team.

WP7 leader will be responsible for the validation and evaluation process, supported by Working Group 8 (Implementation Team).

As far as possible, testing frameworks and automatic verification tools will be used for testing the single components. Test automation is expensive and especially difficult for evaluation purpose, so it has to be considered as an addition, not a replacement, to manual testing.

While the validation and evaluation of the single components can be carried out by the responsible partner of each component, the evaluation of the full platform will be performed in the planned progress meetings.

Regression testing is intended to be performed periodically, after upgrades and changes to components and the overall platform are done. “Factory” and “magazine” tests will be repeated to ensure that no bugs are introduced. As far as possible, this regression testing will be done in an automated way, by using test frameworks.

## 4 Evaluation and Validation Environment

This section describes the hardware & software configuration, data sets and other means needed to perform the tests.

Remark: Specific needs required for particular Test Cases shall be identified in the related section of the Test Cases.

### 4.1 ARGUGRID Scenarios

The scenarios defined in document D.1.2 will be used during the entire evaluation and validation plan. All the Use Cases and Test Cases, from low-level component testing to high level System test, will be based on partial or whole scenarios and use cases.

As mentioned in D.1.2, for ease of reference, the scenario cases will be referenced by code names as in the following table:

<i>Scenario</i>	eProcurement scenario		Earth Observation scenario		Business Planning & Outsourcing scenario
<i>case</i>	Request For Proposals case	Optimizing eAuctions parameters case	Service selection case(oil spill)	Service orchestration case (fire detection)	Computer assembly investment case
<i>case code name</i>	<b>eProcRFP</b>	<b>eProcOpt</b>	<b>EOSelect</b>	<b>EOComp</b>	<b>Invest</b>

**Table 6: Code names for the scenario cases**

The first eProcurement use case, **eProcRFP**, explores a Request for Proposal issued by a Buyer organisation, the respective responses from two consortia of Suppliers -where each consortium consists of two (2) Suppliers and is in fact a Virtual Organization- and the evaluation of the offers by the Buyer, using various criteria and arguments.

The second eProcurement use case, **eProcOpt**, explores the issue of optimising the Business-to-Business reverse eAuction setup parameters, so that a Buyer organisation can organize the best possible eAuction for his interests. In order to do that though, the Buyer - or an eAuction services consultant - should estimate various factors and parameters related to the market and the item to procure, using various criteria and arguments.

The two **Earth Observation** eBusiness use cases will explore the business environment where the Earth Observation service and data providers are currently offering their products. This environment is somehow chaotic and decentralised and it seems to us that argumentation and agents could provide a valuable solution without introducing too many modifications in the current picture.

And finally, the **Invest** case investigates the development of formal frameworks for modelling contracts, contracts negotiation and conflict resolution that are essential in the business process for outsourcing activities.

#### 4.1.1 A brief description of the “RFP case” (“eProcRFP”)

The RFP case (Request for Proposals) refers to the procedure for procuring any complex system, project or service from a supplier, or from a consortium of suppliers (that could form a VO). The case described in this section includes inviting possible suppliers, selecting two of them, asking for specific offers from these two suppliers, and evaluating the offers according to specific criteria and arguments.

The evaluation criteria can be rather “vague” and subjective, as the buyer does not have (nor does he need in fact) the expertise that is required in order to identify quantifiable characteristics in the

proposed solutions. As a result, the evaluation procedure can be described as heavily subjective and argumentation-based.

For this use case, a specific RFP case has been proposed: The RFP for procurement of an e-Ordering system, along with a high-speed Internet connection. A very detailed description of this case, along with a case run with actual and realistic data, can be found in deliverable D.1.2, chapter 3.

Concretely, following D.1.2, we will assume that the RFP is described in terms of the following features

<i>Id</i>	Feature description
<i>f1</i>	the system provides self-service supplier catalogue processing
<i>f2</i>	the system provides catalogue changes tracking and authorisation mechanisms
<i>f3</i>	the system supports shopping cart aggregation
<i>f4</i>	the offer includes integration of the e-Ordering system with the buyer company's enterprise resource planning system (ERP)
<i>f5</i>	the system supports out-of-catalogue requests
<i>f6</i>	the system supports 'round-trip' / 'punch-out' requests
<i>f7</i>	the system supports multiple framework agreements
<i>f8</i>	the system supports multiple and concurrent product categorisations
<i>f9</i>	the system supports multiple workflow types
<i>f10</i>	the offer is based on a 3-year flat contract cost
<i>f11</i>	the system supports electronic auctions

The buyer wishes to achieve three general goals, namely:

- To decrease his direct or indirect purchasing and inventory costs.
- To achieve better control on its spending, reduce off-contract spending, gain insight on purchasing patterns, etc.
- To enhance the standardisation and automation of buyer-supplier communications.

These goals may only be achieved partially by offers; we therefore consider gradations of them. These gradations constitute the potential benefits of offers. We use the following twelve benefits:

A) Decrease in costs:

1. high\_decrease\_costs
2. average\_decrease\_costs
3. fair\_decrease\_costs
4. poor\_decrease\_costs

B) Improvement of spending control

5. high\_spending\_control
6. average\_spending\_control
7. fair\_spending\_control
8. poor\_spending\_control

C) Enhancement of buyer-supplier communication

9. high\_enhancement
10. average\_enhancement
11. fair\_enhancement
12. poor\_enhancement

We will assume that there are 12 suppliers denoted  $s_1, s_2 \dots s_{12}$ , each making an offer. These offers correspond to different e-Ordering systems. Each offer is defined by 11 features for the proposed

eOrdering system and a selling price. Concretely, we use the following offers for each respective supplier:

<i>id</i>	<i>Features offered</i>	<i>price</i>
s <sub>1</sub>	f1, f5, f11	12.3Keuros
s <sub>2</sub>	f2, f4	11.2Keuros
s <sub>3</sub>	f1, f3, f5, f7, f9, f10, f11	17.0Keuros
s <sub>4</sub>	f2, f5, f10, f11	12.8Keuros
s <sub>5</sub>	f7, f10	9.9Keuros
s <sub>6</sub>	f1, f10	10.8Keuros
s <sub>7</sub>	f2, f4, f9	13.1Keuros
s <sub>8</sub>	f1, f3, f4, f10, f11	14.2Keuros
s <sub>9</sub>	f2, f5	11.7Keuros
s <sub>10</sub>	f2, f3, f4, f5, f8, f10	12.1Keuros
s <sub>11</sub>	f1, f2, f4, f5, f7, f9, f11	19.9Keuros
s <sub>12</sub>	f1, f2, f3, f4, f5, f7, f9, f10, f11	22.5Keuros

Note that this case will be tested for CaSAPI (see Table 5).

#### 4.1.2 A brief description of the “Optimising eAuction parameters case” (“eProcOpt”)

Industrial procurement is a complex problem that requires expertise in the domain of the product/service to be purchased. Experts are not only required for understanding the various features of items but may also help the buyer identify the market characteristics. Once a clear representation of the procurement problem has been obtained, it is time to start the procurement process. In this case we focus on the following question of such a process: “Should an electronic-auction (eAuction) be performed for this item?”

The OGC (Office of Government Commerce, UK) has developed a simple and free electronic tool (<http://www.ogc.gov.uk/>), based on MS-Excel ([4]) to provide assistance in evaluating, whether an eAuction is suitable for a specific procurement. To reach a decision the user is asked to make assumptions and provide input at various decision points. For example the user is asked to rate and input the complexity of the procurement, the completeness, the accuracy of specifications, etc.

Having decided that an eAuction is appropriate, a user can use a cosmoONE tool, (for full details, see D.1.1, section 3.4.2) in flowchart format, which, given specific assumptions and inputs, will propose/select the best type of eAuction, and then it will suggest the best parameter values for the selected type.

The main interest (and difficulty) of this case is in helping the buyer to make the correct decisions about whether an eAuction is an appropriate tool and the assumptions and possible inputs to the cosmoONE tool to decide which type of eAuction is the most suitable.

We consider the real-world problem of the company cosmoONE, the consultant, which provides assistance to the client in evaluating, whether an electronic-auction (eAuction for short) is suitable for a specific procurement. We consider the specific case of the IT department of a large organization wishes to select a fleet management system for its 2500 cars and trucks. cosmoONE knows how to argue the case. cosmoONE, as a consultant, uses inputs from the client, his own information and knowledge regarding the market and the suppliers, applies argumentation to make decisions. If the data provided by the client, combined with the consultants data, are still not enough to decide, then the consultant requests the missing data from a market analytics service provider.

A very detailed description of this case, along with a case run with actual and realistic data, can be found in D.1.2, section 4. For the purposes of evaluation, we will assume that the consultant needs to evaluate, for the specific fleet management system for 2500 cars and trucks:

- the competitive environment, by answering the following questions:
  1. What is the level of market competition for this procurement?
  2. What is the predicted market value of this procurement?
- the procurement specification, by answering the following questions:
  3. What is the level of service complexity for this procurement?
  4. With what level of accuracy can the procurement be specified?
- the strategic value of the item, by answering the following questions:
  5. What level of supply market complexity is there in this procurement?
  6. What level of strategic importance is attached to this procurement?
- the nature of procurement award criteria, by answering the following questions:
  7. To what extent are the procurement award criteria quantifiable?

In our tests, we will assume that the business knowledge of the consultant amounts to providing the following answers

1. don't know
2. don't know
3. the level is high since the cars are spread all over the country
4. the level is low since the requirements are not clear enough
5. don't know
6. don't know
7. don't know

Note that for testing purposes we focus on the initial step of the original case described in D.1.2, namely the step of assessing whether an eAuction is at all suitable, using ARGUGRID technology to model the consultant instead of the OGC tool currently deployed by cosmoONE.

Note that this case will be tested for MARGO only (see Table 5).

#### **4.1.3 A brief description of the “Service Selection case” (“EOSelect”)**

Selection of the fittest Image Provider is a complex task. The number of possible choices, the different providers' characteristics, and the deep knowledge required for selecting valid providers, makes this use case interesting enough for our evaluation.

In this use case, a wide number of web services acting as Image Providers will be available, each one with different characteristics. The difficulty will be in selecting the provider that fits best the preferences of the user.

Five different image providers have been created for this scenario (described in Table 7). Every image provider can work in different modes, and depending on the selected mode, some image acquisitions will be possible, others not. Properties of these images (covered area, resolution...) will depend also on the selected mode.

Providers have different characteristics and the preferred one would depend on users' preferences. Moreover, a provider considered a-priori as a bad option, could be the only one to provide a valid image in certain conditions.

Selecting the fittest image for one specific user would require analysing all the available images that could be provided by all the image providers. This task requires a lot of time and it is done currently in a manual way, often involving e-mail and telephone communications.

Expertise is also required to select a valid image for a specific purpose, as different information can be extracted from images with different properties. The fittest image has to be chosen among the images that can be used for a specific purpose.

Three different actors can be considered in this scenario, and they are described in section 4.3.

Note that this case will be tested for the overall platform (see Table 5).

#### **4.1.4 A brief description of the “Service Orchestration case” (“EOComp”)**

The second EO use case starts from the previous use case, and goes one step beyond. For normal, non-technical users, creating a tailored product from the existing image providers and service providers is a complex and difficult task.

For creating such products, a deep knowledge of the existing services and their characteristics is required. Besides, orchestrating such services into a valid workflow able to create a tailored product is also a complex task.

Fifteen different image processing services have been created (described in section 4.4.2). They can be combined in different ways to produce a final product for a specific area. After selecting the fittest image, some clipping could be required in order to reduce the processing time of the final. It could be also needed to convert the image from one format to another, in order to use some processing service with limited input or output formats.

In this use case, a “fire detection” workflow will be orchestrated in order to create a “fire product” over an area of interest for a user. The different steps required for getting such a product, will be done automatically. Starting from selecting the fittest image, performing any required pre-processing of the image, and creating finally the desired product.

The same three actors used in the previous scenario “EOSelect” will be used in this one, and different results are expected accordingly to the preferences of every one of them.

Note that this case will be tested for the overall platform (see Table 5).

#### **4.1.5 A brief description of the “Computer Assembly Investment case” (“Invest”)**

This use case is about supporting business planning and outsourcing to new countries, and in particular supporting the choice of locations and business plan. This scenario is mostly intended to guide the theoretical developments in WP4, but a concrete knowledge base for supporting the porting of this case onto the ARGUGRID platform has been given in deliverable D.4.1. Further developments on this formalisation may allow for specific tests on this case to be performed in the third year.

Note that none of the currently planned tests in Table 5 involve this case. We may consider tests for this case in the third year of the project, if time allows.

## **4.2 ARGUGRID Integration Platform**

The following items have been identified to be part of the testing environment:

### **4.2.1 EO Server**

A PC will provide access to the web services developed by GMV. The full set of web services will be hosted on this machine, and a database will contain all the information for the different web services.

### **4.2.2 GRIA nodes**

The services provided by the GMV industrial partner, will be deployed in GRIA Grid nodes, i.e. dedicated machines running the GRIA Grid middleware, in the ICCS facilities. Although, the definite numbers of PCs used for the evaluation and the experimentation of the ARGUGRID platform, has not been yet finalized, we are planning to use as many as four or five different machines.

The evaluation of the Grid component of the platform will not be emulated but rather be tested in a real Grid environment, as decided in D.6.3 deliverable

#### 4.2.3 GOLEM containers

A set of machines, between four and eight, running one or more GOLEM containers will be needed. Every GOLEM container will hold one or more agents.

#### 4.2.4 KDE server

This computer will run the KDE environment. The user will use it as a client interface.

### 4.3 Actors

In order to provide a detailed description of the tests, we give here a concrete description of the actors used, providing concrete instances. For a justification of the concrete choices made here we refer to deliverable D.1.2.

This section details the identified actors that interact with components or with the whole system:

- Administrator user: User that would configure or instruct the platform in order to set up specific configurations or to transfer knowledge and rules to the platform.
- EORequestors: Users that require EO products. Needs for these users can range from selecting the fittest image provider to creating complex products in which several services have to be orchestrated. These are the defined EORequestors:
  - **EORequestor-Crisis:** Main concern for this user is delivery time. Typical user in a crisis event that needs information as soon as possible. Quality of the final product is not important as long as it meets some minimum levels. Price is not important.
  - **EORequestor-Research:** Main concern is quality of the final product. Price is in a secondary level but also important. Delivery time is not very important. This profile matches a research user.
  - **EORequestor-Private:** Price is the most important issue. This user has daily needs for products, or he has to perform a lot of operations. Fulfilling certain minimum requirements, time and quality are not important.
- ImageProviders: These actors provide EO images. There are five different web services acting as EO image providers. They are described with more detail in section 4.4.1.
- ImageProcessors: These actors process EO products in order to create other EO products. The web services that act as these actors are described with more detail in section 4.4.2.
- Buyer-I: The Buyer is a requestor who issues an RFP, regarding the purchasing of an eOrdering system. The RFP describes expected benefits and a few quantifiable features. The goal of this actor is to select the most advantageous offer for his request and award the contract to the respective Supplier or consortium of suppliers.
- A-type suppliers: The Suppliers provide solutions based on products and/or services. A-type suppliers are supplying eOrdering systems and applications. Their goal of these actors is to respond to the Buyer's RFP, in order to win the deal and get the contract from the I-Buyer.
- B-type suppliers: The Suppliers provide solutions based on products and/or services. B-type suppliers are Internet Service providers. Their goal is to combine their forces with an A-type supplier to form a consortium to respond to the Buyer's RFP, in order to win the deal and get the contract from the Buyer.
- Buyer-II: He wants to determine if an eAuction is appropriate for a specific procurement need and which one. His goal is to determine the relevance of adopting an eAuction, in order to get the best deal from the suppliers.

- eAuction Consultant: He will evaluate the benefits of an eAuction with respect to the product and the market. If the data provided by Buyer II, combined with the consultant’s data, are still not enough to decide, then the consultant request the “missing” data from a market analytics service provider.
- Market Analyst: He collects data, performs market research and builds know-how in specific market sectors, regarding suppliers, products, services, prices, trends etc, in order to provide requestors with market intelligence.

#### 4.4 Web services

As using real EO information providers for testing purpose is not feasible, several Web Services have been created to simulate the EO providers. These Web Services can be classified in two groups:

##### 4.4.1 Image providers

They provide the images that will be used as inputs in the workflows. Every service has different characteristics, and the whole set of web services will offer a big number of combinations. The characteristics used in our examples are:

- Price: Price of getting an image from this service.
- Resolution: Providers can offer images with different resolutions, ranging from 1 to 100 meters.
- Image size: This characteristic can be independent from resolution. Different providers offer images with different sizes.
- Image type: Possible images types have been simplified to OPTICAL and RADAR images
- Acquisition time: Time and date of a possible image acquisition.
- Delivery time: Time and date when an acquired image could be available.
- Image availability: Depending on the Web Service, different sets of available images have been created, making the selection task more complicated.

Five web services with different characteristics have been created. The following Table 7 shows the list of web services with their main characteristics. The tree last characteristics (acquisition and delivery time, and availability) are summarized into the number of available images. It has to be taken into account that in some cases, a provider with less availability can provide the fittest image for a specific image requestor.

WS name	Price	Resolutions	Image size	Image type	Available images
eocatalogue1	150	10, 25, 50m	200, 400, 1000 km	OPTICAL	9
eocatalogue2	150	10, 25, 50m	200, 400, 1000 km	RADAR	9
eocatalogue3	200	5, 10m	50,100 km	OPTICAL & RADAR	30
eocatalogue4	150	1, 5, 10m	10,50,100 km	OPTICAL & RADAR	28
eocatalogue5	250	1,5,10,25,50m	10,50,100,250,500 km	OPTICAL & RADAR	10

**Table 7: Image provider service's characteristics**

##### 4.4.2 Image processing services

Image processing services: These providers will process images in order to extract information, transform the image, or create new images. Several basic functions have been identified, and several web services of each type have been created:

- **Format Converter:** Transforming an image into another format can be necessary before calling another processing service. Several web services have been created with different characteristics: price, accepted input formats, available output formats, and processing time.

The following Table 8 summarizes the properties of every format converter:

WS name	Price	Deliv.time	Input formats	Output formats
formatConverter1	50	1800'	GEOTIFF,JPG2000	GEOTIFF,JPG2000
formatConverter2	200	900'	GEOTIFF,JPG2000	GEOTIFF,JPG2000
formatConverter3	300	300'	GEOTIFF,JPG2000,HDF,SHAPE	GEOTIFF,JPG2000,HDF
formatConverter4	100	1800'	GEOTIFF,JPG2000,HDF,SHAPE	GEOTIFF,JPG2000
formatConverter5	300	1800'	GEOTIFF,JPG2000,HDF	GEOTIFF,JPG2000

**Table 8: Format converter service's characteristics**

- **Clipping:** Some EO processing tasks are very dependent on image's size, and decreasing the size by clipping can speed up the process. Several web services have been created to simulate a wide offer of such services. Characteristics of these services will have different values for price, accepted input formats, available output formats, and processing time.

The following Table 9 summarizes the properties of every clipping service:

WS name	Price	Deliv.time	Input formats	Output formats
clipping1	50	1800'	GEOTIFF,JPG2000	GEOTIFF
clipping2	200	900'	GEOTIFF,JPG2000,HDF	GEOTIFF,JPG2000,HDF
clipping3	300	300'	GEOTIFF,JPG2000,HDF,SHAPE	GEOTIFF,JPG2000,HDF,SHAPE
clipping4	100	1800'	GEOTIFF,JPG2000,HDF,SHAPE	GEOTIFF,JPG2000,HDF,SHAPE
clipping5	300	500'	GEOTIFF	GEOTIFF

**Table 9: Clipping service's characteristics**

- **Fire detection:** Detecting possible fires in an optical image is a further example of an Image Processing Service. A set of these services with different characteristics is available for testing purposes.

The following Table 10 summarizes the properties of every fire detection service:

WS name	Price	Deliv.time	Input formats	Output formats	Quality <sup>4</sup>
fire-detection1	50	1800'	GEOTIFF, JPG2000	GEOTIFF	Medium(60)
fire-detection2	200	900'	GEOTIFF, JPG2000, HDF	GEOTIFF, JPG2000, HDF	Poor (40)
fire-detection3	300	300'	GEOTIFF, JPG2000, HDF, SHAPE	GEOTIFF, JPG2000, HDF, SHAPE	High(85)
fire-detection4	100	1800'	GEOTIFF, JPG2000, HDF, SHAPE	GEOTIFF, JPG2000, HDF, SHAPE	Excellent(95)
fire-detection5	300	500'	GEOTIFF	GEOTIFF	Medium(50)

**Table 10: Fire detection service's characteristics**

<sup>4</sup> Evaluating the quality of image processing is a complex task, involving several measurements as minimum threshold for detection, or false positive and false negative rates. For simplification, here we consider values between 0 (worst) and 100 (perfect) to simulate mean processing quality.

All image processing services have the following common characteristics:

- Price: Price of using the service.
- Delivery time: Time needed for processing and delivering an image. This value has been simplified to a “mean processing time” for simplifying our scenarios.
- Input formats: List of accepted input image’s formats.
- Output formats: Valid output formats.

Quality value (see footnote 4) can be provided to simulate mean quality of a service provider.

#### **4.5 Other components**

Some of the components that form the ARGUGRID platform require a specific environment, or third party components. This section describes such components and their special requirements:

- MARGO (using CaSAPI) [1] supports the reasoning and decision-making of argumentative agents performing service selection and composition in ARGUGRID. Both of MARGO and CaSAPI are platform-independent but they require a Prolog installation. We have decided to adopt both SWI-Prolog 5.6.x and SICSTUS Prolog 3.x for implementing and testing MARGO during its building process, and SICSTUS Prolog 3.x for implementing and testing CaSAPI.
- GOLEM: within the ARGUGRID project, we require Java 1.6+ to run a GOLEM node. GOLEM has the same requirements as MARGO with respect to Prolog.

## 5 Test Cases

This section describes the full list of Test Cases designed for validating and evaluating the platform.

Section 5.1 summarizes the Test Cases grouped by their type: “magazine” tests and “factory” tests. Section 5.2 presents the “magazine” tests used for evaluation, and following sections 5.3, 5.4 and 5.5 present the full list of “factory” Test Cases following the bottom-up approach: Components, Integration of components, and the ARGUGRID platform.

### 5.1 Classification of Test Cases

In order to clarify the different aspects of the system that will be validated, two lists with Test Cases are provided below. The first list summarizes the “magazine” tests that will be used for evaluating the platform, while the second list shows the “factory” tests cases that will allow us to validate the platform.

#### 5.1.1 “Magazine” tests

This list shows the “magazine” tests that will be used for evaluating the platform. Beside some test at platform level and integration level, five tests at component level are included, in order to evaluate the Argumentation engine, formed by the CaSAPI and MARGO components.

TEST CASE IDENTIFICATION	DESCRIPTION
<b>COMPONENT TESTING</b>	
<b>MT.CaSAPI.SelectDominantSuppliers</b>	Suppliers in a “dominant” position will be identified from a list of suppliers (eProcRFP). Market segment: B2B/cosmoONE Criteria: QoR for DM, AL for DM, CoK for DM
<b>MT.CaSAPI.RankDominantSuppliers</b>	Suppliers in a “dominant” position will be measured against one another from a list of suppliers (eProcRFP). Market segment: B2B/cosmoONE Criteria: QoR for DM, AL for DM, CoK for DM
<b>MT.MARGO.DecideJustifyEAuction</b>	MARGO makes a decision about suitability of an eAuction. (eProcOpt) Market segment: B2B/cosmoONE Criteria: QoR for DM, AL for DM, CoK for DM
<b>MT.MARGO.SelectJustifyService</b>	MARGO makes a decision based on EO knowledge and user preferences. (EOSelect) Market segment: B2B/GMV Criteria: QoR for DM, AL for DM, CoK for DM
<b>MT.MARGO.ServiceComposition</b>	MARGO makes a decision based on EO knowledge and user preferences. (EOComp) Market segment: B2B/GMV Criteria: QoR for DM, AL for DM, CoK for DM.
<b>COMPONENTS INTEGRATION TESTING</b>	
<b>MT.PLATON-GOLEM.Scalability</b>	GOLEM agents look for partners to form a VO, using the PLATON P2P platform to discover them. The PLATON environment fails to discover an agent and the requester informs the user of the failure. Market segment: existing systems Criteria: Scalability

<b>ARGUGRID PLATFORM TESTING</b>	
<b>MT.EOSelect.ContractNegotiationMCP</b>	Selecting the fastest image provider for an Oil Detection and negotiating the price. Market segment: B2B/GMV and InforSense Criteria: QoR for DM, QoR for NC, AL for DM, AL for CN.
<b>MT.EOSelect.WfContractNegotiationMCPR</b>	Selecting the highest-quality/best price image provider for an Oil Detection and negotiating quality and price. Market segment: B2B/GMV and InforSense Criteria: QoR for DM, QoR for NW, AL for DM, AL for NW
<b>MT.EOComp.ContractNegotiationMCP</b>	Selecting the fastest product for fire detection and negotiating the price. Market segment: B2B/GMV and InforSense Criteria: QoR for DM, QoR for NC, AL for DM, AL for NC
<b>MT.EOComp.WfContractNegotiationMCPR</b>	Selecting the highest-quality/best product for fire detection and negotiating quality and price. Market segment: B2B/GMV and InforSense Criteria: QoR for DM, QoR for NC, AL for DM, AL for NC

This list is by no means exhaustive and may be modified in later deliverable D.7.3, if some functionalities of the system are not completely covered by the current list and in line with the developments of the project.

### 5.1.2 “Factory” tests

This list shows the fifteen tests designed for validating the ARGUGRID components, the ten tests designed for validating the components integration, and the test designed for validating the whole platform with an end-to-end Test Case.

<b>TEST CASE IDENTIFICATION</b>	<b>DESCRIPTION</b>
<b>COMPONENT TESTING</b>	
<b>FT_C.GRIA.ExecuteGRIAServices</b>	Find a GRIA service, with a valid SLA, and executes the service.
<b>FT_C.GRIA.Registry</b>	Set up the GRIA registry, store SLA templates and XML files and link them with GRIA services.
<b>FT_C.PLATON.Routing</b>	PLATON resolves a request for a 2-dimensional query.
<b>FT_C.PLATON.LoadBalancing</b>	PLATON re-distributes a given load change when a peer adds new resources.
<b>FT_C.KDE.Authoring</b>	Authoring, annotating and saving abstract workflows.
<b>FT_C.KDE.Argubroker.DPML_to_Prolog</b>	Translation from InforSense DPML workflow in XML format to Prolog format
<b>FT_C.KDE.Argubroker.WSMO_to_Prolog</b>	Translation from WSMO in XML format to Prolog format
<b>FT_C.KDE.Argubroker.WSML_to_Prolog</b>	Translation from WSML in XML format to Prolog format
<b>FT_C.KDE.Argubroker.Prolog_to_DPML</b>	Translation from concrete DPML workflow in Prolog format to executable DPML workflow in XML format.
<b>FT_C.KDE.Registry</b>	Publish and retrieve semantic service descriptions.
<b>FT_C.KDE.Execution.1</b>	Set-up of concrete workflow for execution.
<b>FT_C.KDE.Execution.2</b>	Execution of executable workflow.
<b>FT_C.GOLEM.AgentDeployment</b>	A new agent is started within a container with all the

	associated knowledge bases and reasoning abilities, sensors and effectors. Other agents are capable to contact the new agent dynamically without intervention of the human users or even the agent's own actions.
<b>FT_C.GOLEM.ObjectDeployment</b>	A new object is started within a container. Agents are capable to view and inspect its state and functionality without intervention of the human users.
<b>FT_C.GOLEM.Interactions</b>	Two or more agents communicate in the agent environment using objects. Agents communicate using an object serving as a protocol validator for their interaction.
<b>INTEGRATION TESTING</b>	
<b>FT_I.PLATON-GRIA.Discovery</b>	PLATON resolves a 2-dimensional query for a GRIA service.
<b>FT_I.PLATON-GOLEM.DiscSuccess</b>	GOLEM agents look for partners to form a VO, using the PLATON P2P platform to discover them.
<b>FT_I.PLATON-GOLEM.DiscFailure</b>	GOLEM agents look for partners to form a VO, using the PLATON P2P platform to discover them. The PLATON environment fails to discover an agent and the requester informs the user of the failure.
<b>FT_I.GRIA-KDE.Connectivity</b>	KDE client find GRIA services, with valid SLA templates, and executes them.
<b>FT_I.GRIA-KDE.SLARetrieval</b>	KDE registry retrieves SLA template associated with GRIA services.
<b>FT_I.GOLEM-KDE.WorkflowCreation</b>	An abstract workflow is created.
<b>FT_I.GOLEM-KDE.ConcreteWorkflow</b>	Solving an abstract workflow into a concrete workflow.
<b>FT_I.GOLEM-KDE.PartialWorkflow</b>	Solving a partial workflow.
<b>FT_I.GOLEM-KDE.Semantic</b>	Agent queries Semantic registry.
<b>FT_I.GOLEM-MARGO.Service</b>	A service requester selects a concrete service.
<b>PLATFORM TESTING</b>	
<b>FT_I.ARGUGRID. EndToEndConsumerScenario</b>	End to end scenario.

## 5.2 “Magazine” tests: ARGUGRID Evaluation

“Magazine” tests focus on the added value functionalities of the ARGUGRID platform, and will allow us to evaluate the platform.

In order to evaluate the “Decision-Making” capabilities of the platform, two components are going to be evaluated: CaSAPI and MARGO, in the context of the e-procurement scenario and the EO scenario. For a more general evaluation of the platform, the EO scenario will be used.

### 5.2.1 CaSAPI component

#### Test Case MT.CaSAPI.SelectDominantSuppliers

This Test Case allows characterising suppliers that are in a “dominant” position with respect to the other competing suppliers, in other words these suppliers are overall preferred to the others (because of the characteristics of their offers). Such suppliers do not always exist, but when they do exist, it is important to identify them, as these will provide without doubt, the best offers, independently of the buyer's order of preferences. The suppliers in dominant position will be found using CaSAPI, for a suitable representation of the business knowledge underlying the RFP scenario, that has been provided in section 4.1.1. The outcomes of CaSAPI will be compared against the outcome of the methods used within the e-procurement industry, as deployed by CosmoONE.

<b>Test Case Id</b>	<b>MT.CaSAPI.SelectDominantSuppliers</b>	
<b>Summary</b>	Suppliers in a “dominant” position will be identified from a list of suppliers (eProcRFP).	
<b>Actors</b>	Buyer-I, A-type suppliers, CaSAPI	
<b>Preconditions</b>	Medium-size benchmark of fictitious and real suppliers in e-procurement.	
<b>Triggers</b>	Buyer-I has received several offers by a number of A-type suppliers.	
<b>Steps</b>	<b>Step</b>	<b>Result</b>
	1. – Load user preferences/business knowledge into CaSAPI tool.	Information loaded ok.
	2. – Launching CaSAPI tool.	List of dominant suppliers returned.
<b>Postconditions</b>	Suppliers identified match the ones identified by normal techniques.	
<b>Evaluation criteria</b>	<b>Market: B2B/cosmoONE</b> <b>QoR for DM:</b> The industrial method leads to the choice of the offer from supplier $s_{12}$ only (see section 4.1.1 for details about eProcRFP scenario). This result will be compared against the result provided by dominance method, and any difference will be evaluated by cosmoONE experts	
	<b>Market: B2B/cosmoONE</b> <b>AL for DM:</b> The typical time required for providing the ranking by cosmoONE will be compared with the time required by our solution.	
	<b>Market: B2B/cosmoONE</b> <b>CoK for DM:</b> The inputs required from the Buyer-I user using the industrial method and CaSAPI will be compared. Moreover, both solutions will be evaluated taking into account how the needed inputs increase with the size of the problem	
<b>Notes</b>		
<b>Author, date &amp; version</b>	Paul Matt, Francesca Toni, Thanassis Stournaras, 2008/05/19, v1	

### Test Case MT.CaSAPI.RankDominantSuppliers

This second Test Case constitutes a numerical generalisation of the first one. We foresee that we will need an extension of CaSAPI (referred to below as CaSAPI+) to deal with this test. We envisage that CaSAPI+ will perform a counting of arguments and measure the degrees of admissibility or "extent" to which a supplier fulfils the property of dominance.

<b>Test Case Id</b>	<b>MT.CaSAPI.RankDominantSuppliers</b>	
<b>Summary</b>	Suppliers in a “dominant” position will be measured against one other. (eProcRFP)	
<b>Actors</b>	Buyer-I, A-type suppliers, CaSAPI+	
<b>Preconditions</b>	Medium-size benchmark of fictitious and real suppliers in e-procurement.	
<b>Triggers</b>	The Buyer-I has received several offers by a number of A-type suppliers.	
<b>Steps</b>	<b>Step</b>	<b>Result</b>
	1. – Load user preferences/business knowledge into CaSAPI+ tool.	Information loaded ok.
	2. – Launching CaSAPI+ tool.	Evaluation of dominant suppliers returned.
<b>Postconditions</b>	A ranking of suppliers will be returned that matches or improves upon the ranking obtained by conventional techniques.	
<b>Evaluation criteria</b>	<b>Market: B2B/cosmoONE</b> <b>QoS for DM:</b> The ranking provided by the industrial method (see earlier test) will be compared with the ranking obtained by CaSAPI+	
	<b>Market: B2B/cosmoONE</b> <b>AL for DM:</b> The typical time required for providing the ranking by cosmoONE (see earlier test) will be compared with the time required by our solution.	

	<b>Market: B2B/cosmoONE</b> <b>CoK for DM:</b> The inputs required from the Buyer-I user using the industrial method and CaSAPI+ will be compared. Moreover, both solutions will be evaluated taking into account how the needed inputs increase with the size of the problem.
<b>Notes</b>	
<b>Author, date &amp; version</b>	Paul-Amaury Matt, Francesca Toni, Thanassis Stourmaras, 2008/05/19, v1

## 5.2.2 MARGO component

Developed for the ARGUGRID project, MARGO implements an assumption-based argumentation mechanism for decision making. MARGO evaluates the possible decisions, make assumptions if it is required, suggests some solutions, and provides an interactive and intelligible explanation of the choice made.

As a stand-alone tool, we are concerned with the correctness/clarity of the decision/justification/assumptions.

### Test Case MT.MARGO.DecideJustifyEAuction

Based on **eProcOpt**, in this Test Case MARGO decides and justifies whether an eAuction is suitable for a specific procurement.

The input for this Test Case is the possible incomplete knowledge about the market and the products. The expected output is the decision about if an eAuction is suitable for a specific procurement, together with the justification for the decision.

The output is intended to be compared with the ones from the electronic tool based on MS-Excel developed by the Office of Government Commerce (OGC), UK. Contrary to MARGO, the OGC tool cannot make assumptions over the missing information. However, the outputs should be in conformance with the OGC tool when the same assumptions are used for both tools.

<b>Test Case Id</b>	<b>MT.MARGO.DecideJustifyEAuction</b>	
<b>Summary</b>	MARGO makes a decision about suitability of an eAuction. (eProcOpt)	
<b>Actors</b>	eAuction consultant, Buyer-II, MARGO.	
<b>Preconditions</b>	A valid representation about the market, user needs and knowledge.	
<b>Triggers</b>	The Buyer-II requests the eAuction consultant whether an eAuction is suitable and asks for an explanation.	
<b>Steps</b>	<b>Step</b>	<b>Result</b>
	1. – Load user preferences/information into MARGO tool.	Information loaded ok.
	2. – Launching MARGO tool.	Choice is returned.
	3. – Clarifications required.	Justification is returned.
<b>Postconditions</b>	A decision about suitability is taken. A justification of the decision is provided.	
<b>Evaluation criteria</b>	<b>Market: B2B/cosmoONE</b> <b>QoR for DM:</b> MARGO allows to decide to adopt (or not) an eAuction. When the OGC tool is applicable, the result is comparable to the result provided by the OGC tool.	
	<b>Market: B2B/cosmoONE</b> <b>CoK for DM:</b> MARGO requires less information than the OGC tool.	
	<b>Market: B2B/cosmoONE</b> <b>AL for DM:</b> Collecting knowledge of experts about the product and the market is time consuming. MARGO requires less time in comparison with the OGC tool to perform the same decision.	
<b>Notes</b>		

<b>Author, date &amp; version</b>	Maxime Morge, 2008/04/26, v1
-----------------------------------	------------------------------

### Test Case MT.MARGO.SelectJustifyService

This Test Case is based on **EOSelect**. MARGO selects a satellite in accordance to the preferences of the requester and the possible incomplete knowledge of the provider.

The input for this Test Case is the possible incomplete knowledge about the specifications of the satellites, and the preferences of the satellite. The expected output is the satellite service selection and the justification for that selection.

No other software application can be used for comparing the results; therefore the expert user evaluation (GMV) will be used.

<b>Test Case Id</b>	<b>MT.MARGO.SelectJustifyService</b>	
<b>Summary</b>	MARGO makes a decision based on EO knowledge and user preferences. (EOSelect)	
<b>Actors</b>	EORequestor, ImageProviders, MARGO.	
<b>Preconditions</b>	A valid representation of user needs and knowledge.	
<b>Triggers</b>	The user requests a service selection and asks for a justification.	
<b>Steps</b>	<b>Step</b>	<b>Result</b>
	1. – Load user preferences/information into MARGO tool.	Information loaded ok.
	2. – Launching MARGO tool.	Choice is returned.
	3. – Clarifications required.	Justification is returned.
<b>Postconditions</b>	A Satellite provider is selected, and a justification of the selection.	
<b>Evaluation criteria</b>	<b>Market: B2B/GMV</b> <b>QoR for DM:</b> we will compare the solution computed by MARGO with 1) that obtained by GMV and 2) that obtained by the KDE starting from an abstract workflow	
	<b>Market: B2B/GMV</b> <b>AL for DM:</b> we will consider the time saved by using MARGO with respect to that 1) needed by GMV to make the same decision 2) needed by the ARGUGRID platform using KDE for authoring an abstract workflow	
	<b>Market: B2B/GMV</b> <b>CoK for DM:</b> With MARGO, the inputs are the knowledge of the expert about the services and the specific case (e.g. the weather). This knowledge is formalized and embedded in MARGO for selecting services. This will be compared with the amount of information required by a GMV consultant to come up with the same selection.	
<b>Notes</b>		
<b>Author, date &amp; version</b>	Maxime Morge, 2008/04/26, v1	

### Test Case MT.MARGO.ServiceComposition

This Test Case is based on **EOComp**. MARGO composes some services for creating a tailored product in accordance to the preferences of the requester and the possible incomplete knowledge of the provider.

The input for this Test Case is the possible incomplete knowledge about the specifications of the services, and the preferences of the EORequestor. The expected output is the final product.

This Test Case is similar to the process performed by MARGO in MT.EOComp.ServiceOrchestration.1 expected that:

- i) The decision is better due to the using of the expert knowledge which specify that the fire detection processing is decreasing since the size of the image is small and, for this purpose, a clipping service can be used;
- ii) This Test Case requires less effort than its current manual performance by GMV or by starting from the abstract workflow setup by the EORequestor.

No other software application can be used for comparing the results; therefore the expert user evaluation (GMV) will be used.

<b>Test Case Id</b>	<b>MT.MARGO.ServiceComposition</b>	
<b>Summary</b>	MARGO makes a decision based on EO knowledge and user preferences. (EOComp)	
<b>Actors</b>	EORequestor-Crisis, ImageProviders, ImageProcessors, MARGO.	
<b>Preconditions</b>	A valid representation of user needs and knowledge about the existing services.	
<b>Triggers</b>	The user requests services for creating a fire product.	
<b>Steps</b>	<b>Step</b>	<b>Result</b>
	1. – Load user preferences/information into MARGO tool.	Information loaded ok.
	2. – Launching MARGO tool.	Fire product is returned.
<b>Postconditions</b>	A Satellite provider is selected, and a justification of the selection.	
<b>Evaluation criteria</b>	<b>Market: B2B/GMV</b> <b>QoR for DM:</b> we will compare the solution computed by MARGO with 1) that obtained by GMV and 2) that obtained by the KDE starting from an abstract workflow	
	<b>Market: B2B/GMV</b> <b>AL for DM:</b> we will consider the time saved by using MARGO with respect to that 1) needed by GMV to make the same decision 2) needed by the ARGUGRID platform using KDE for authoring an abstract workflow	
	<b>Market: B2B/GMV</b> <b>CoK for DM:</b> With MARGO, the inputs are the knowledge of the expert about the services (e.g. clipping services decrease the size for images) and the specific case (e.g. the weather). This knowledge is formalized and embedded in MARGO for selecting services. This will be compared with the amount of information required by a GMV consultant to come up with the same selection.	
<b>Notes</b>		
<b>Author, date &amp; version</b>	Maxime Morge, 2008/04/26, v1	

### 5.2.3 Component integration

#### Test Case MT.PLATON-GOLEM.Scalability

This test is intended to evaluate the scalability of PLATON and GOLEM.

<b>Test Case Id</b>	<b>MT.PLATON-GOLEM.Scalability</b>	
<b>Summary</b>	The Test Case is about the integration between PLATON and GOLEM. We are going to test the scalability of the overall platform in terms of: number of containers deployed and time to discover a peer in the distributed network.	
<b>Actors</b>	No actors involved; PLATON and GOLEM components used.	
<b>Preconditions</b>	A set of GOLEM containers deployed on top of PLATON inside the PlanetLab facilities.	
<b>Triggers</b>	An agent which tries to discover other agents.	
<b>Steps</b>	<b>Step</b>	<b>Result</b>

	1. – Deploy PLATON bootstrap node	PLATON node correctly deployed.
	2. – Deploy GOLEM Containers	The containers enters the network creating a distributed and balanced K-D Tree
	3. – An agent tries to discover another agent	The agent discovers the partner in a logarithmic number of hops.
<b>Alternative path</b>	<b>Step</b>	<b>Result</b>
	3b. – An agent does not discover a partner	The failure is reported in logarithmic time.
<b>Postconditions</b>	The agents can start a negotiation.	
<b>Evaluation criteria</b>	<b>Market: Existing P2P systems</b> <b>Criteria: Scalability</b>	
<b>Notes</b>		
<b>Author, date &amp; version</b>	Stefano Bromuri, 2008/11/05, v0.1	

## 5.2.4 EO SCENARIO

### Test Case MT.EOSelect.ContractNegotiationMCP

A variant of Use Case EOSelect, described in document D.1.2, and summarized in section 3.1.3 will be used.

A requestor actor will ask for a valid image for detecting oil spills. In this first Test Case, we will consider the EORequestor-Crisis actor, whose main concern is to get a valid image as soon as possible. Price, format or other characteristics are of secondary importance.

The expected output for this Use Case is the selection of the fastest image provider among all the valid and available image providers at the best price. This price is negotiated using the minimal concession protocol (MCP) of WP4 (see D.4.1).

<b>Test Case Id</b>	<b>MT.EOSelect.ContractNegotiationMCP</b>	
<b>Summary</b>	Selecting the fastest image provider for an Oil Detection and negotiating the price.	
<b>Actors</b>	EORequestor-Crisis, ImageProviders, the ARGUGRID platform	
<b>Preconditions</b>	User preferences will match a crisis situation, in which faster time deliveries will be preferred to other considerations.	
<b>Triggers</b>	EORequestor-Crisis initiates session with the ARGUGRID platform.	
<b>Steps</b>	<b>Step</b>	<b>Result</b>
	1. – EORequestor-Crisis asks for a valid image, specifying that this needs to be available as soon as possible.	KDE accepts EORequestor-Crisis request, in the form of an abstract workflow consisting of a single abstract service.
	2. ArguBroker translate request into a MARGO goal.	A MARGO goal.
	3. The MARGO receiving the goal agent (indicated below as EORequestor-Crisis agent) uses GOLEM+PLATON to retrieve suitable service providers with corresponding agents.	A number of providers (each given by a service EPR+agent name) all providing the kind of image requested. These providers are “held” within the mind of the EORequestor-Crisis agent.
	4. MARGO is used by the EORequestor-Crisis agent to select the provider with the fastest delivery time.	A single provider (indicated below as ImProv agent) “held” within the mind of the EORequestor-Crisis agent.
5. The ImProv agent consults ArguBroker about SLA template for the service to be provided to	SLA template	

	EORquestor-Crisis agent. The KDE fetches the SLA templates from GRIA and returns it to the ImProv agent.	
	6. The EORquestor-Crisis agent negotiates with the ImProv agent on price. The EORquestor-Crisis agent starts from the lower-end of the interval offered by the ImProv agent minus some amount. The ImProv agent starts from the upper-end of its allowed interval for price. The agents use the MCP (using their SIM+SDMM). MARGO and GOLEM are involved.	A price.
	7. The EORquestor-Crisis agent returns the EPR of the service represented by the ImProv agent, its details (delivery time etc) as well as the negotiated price to the ArguBroker.	Service description + price for ArguBroker's consumption.
	8. – KDE returns to user proposed image provider, its details, and price.	EORquestor-Crisis visualizes best option characteristics and price.
	9. – EORquestor-Crisis accepts image proposed.	KDE orders image acquisition.
<b>Postconditions</b>	Fittest image acquisition is shown to EORquestor-Crisis at a negotiated price.	
<b>Evaluation criteria</b>	<b>Market: B2B/GMV</b> <b>QoR for DM:</b> The chosen provider is the same that would have been chosen by GMV. <b>QoR for NC:</b> The price is better than the one obtained by GMV.	
	<b>Market: B2B/GMV and InforSense</b> <b>AL for DM:</b> How long is taken to select the best service provider, with respect to InforSense techniques prior to ARGUGRID? <b>AL for CN:</b> How long is taken to generate the contract (price) automatically? How does this compare to the time taken by business experts (GMV)?	
<b>Notes</b>		
<b>Author, date &amp; version</b>	Francesca Toni, Stefano Bromuri, Visara Bromuri, José Barba, 2008/11/20, v2	

### Test Case MT.EOSelect.WfContractNegotiationMCPR

Use Case EOSelect, described in document D.1.2, and summarized in section 3.1.3 will be used.

In this second Test Case, the EORquestor-Research actor is interested in images of high quality, with resolution within given boundaries, and with price being a strong point in its decision. Cheap images will be preferred to expensive ones. A specific price range is specified by the EORquestor-Research actor.

The ideal output for this Test Case is the selection of an image with optimal relation between price and quality, according to the EORquestor-Research preferences (resolution boundaries and price interval). Assume however that no provider exists matching these preferences. The output is then a sub-optimal but sufficiently close match, that the user is presented with nonetheless. This sub-optimal solution is negotiated using the minimal concession protocol with rewards (MCPR) of WP4 (see D.4.1). This sub-optimal solution is the result of negotiation of workflow (in the sense of image quality) and contract (price).

<b>Test Case Id</b>	<b>MT.EOSelect.WfContractNegotiationMCPR</b>
<b>Summary</b>	Selecting the highest-quality/best price image provider for an Oil Detection and negotiating quality and price.
<b>Actors</b>	EORquestor-Research, ImageProviders, the ARGUGRID platform

<b>Preconditions</b>	User preferences amounts to best resolution/price deal, given certain constraints on quality and price.	
<b>Triggers</b>	EORequestor-Research initiates session with the ARGUGRID platform.	
<b>Steps</b>	<b>Step</b>	<b>Result</b>
	1. – EORequestor-Research asks for a valid image, specifying quality boundaries and a price interval	KDE accepts EORequestor-Research request, in the form of an abstract workflow consisting of a single abstract service.
	2. ArguBroker translate request into a MARGO goal.	A MARGO goal.
	3. The MARGO agent receiving the goal (indicated below as EORequestor-Research agent) uses GOLEM+PLATON to retrieve suitable service providers with corresponding agents.	A number of providers (each given by a service EPR+agent name) all providing the kind of image requested. These providers are “held” within the mind of the EORequestor-Research agent. No price matches however.
	4. MARGO is used by the EORequestor-Research agent to select the provider with the closest price to the requested one.	A single provider (indicated below as ImProv agent) “held” within the mind of the EORequestor-Research agent.
	5. The ImProv agent consults ArguBroker about SLA template for the service to be provided to EORequestor-Research agent. The KDE fetches the SLA templates from GRIA and returns it to the ImProv agent.	SLA template
	6. The EORequestor- Research agent negotiates with the ImProv agent on price, using the MCP (using their SIM+SDMM). MARGO and GOLEM are involved. This fails.	Failure.
	7. The ImProv agent starts negotiation with the MCP. Agents again use their SIM+SDMM and MARGO and GOLEM are involved.	An image, of price within the given boundaries but slightly lower quality.
	8. The EORequestor-Research agent returns the EPR of the service represented by the ImProv agent, its details (delivery time etc) as well as the negotiated price to the ArguBroker	Service description + price for ArguBroker’s consumption.
	9. – KDE returns to user proposed image provider, its details, and price.	EORequestor-Crisis visualizes best option characteristics and price.
10. – EORequestor- Research accepts image proposed.	KDE orders image acquisition.	
<b>Postconditions</b>	Fittest image acquisition is shown to EORequestor- Research at a negotiated price and image quality.	
<b>Evaluation criteria</b>	<b>Market: B2B/GMV and InforSense</b>	
	<p><b>QoR for DM:</b> The chosen provider is the same that would have been chosen by GMV.</p> <p><b>QoR for NW:</b> The chosen provider features (workflow) are preferable to the ones obtained by InforSense without the ARGUGRID platform.</p>	
<b>Notes</b>	<b>Market: B2B/GMV and InforSense</b>	
	<p><b>AL for DM:</b> How long is taken to select the best service provider, with respect to InforSense techniques prior to ARGUGRID?</p> <p><b>AL for NW:</b> How long is taken to generate the workflow (image resolution) automatically? How does this compare to the time taken by business experts (GMV)?</p>	
<b>Author, date &amp; version</b>	Francesca Toni, Stefano Bromuri, Visara Bromuri, José Barba, 2008/11/20, v2	

## Test case MT.EOComp.ContractNegotiationMCP

Use Case **EOComp**, described in document D.1.2, and summarized in section 3.1.4 will be used.

A requestor actor will ask for a fire detection product that has to be created from existing Image Providers and Image Processors.

This test is analogous to Test case MT.EOSelect.ContractNegotiationMCP, in that we will consider a crisis situation where the main concern of the EORequestor-Crisis agent is to get a valid product as soon as possible. Price, format or other characteristics are considered as secondary. However, the expected output for this Use Case is a combination of services (an image, a clipping, and a fire detection processing). The combination is obtained from the services that give the required fire detection product as quickly as possible. The prices of all services in the combination are negotiated using the minimal concession protocol (MCP) of WP4 (see D.4.1). In addition to the EORequestor-Crisis agent, this test also involves ImageProvider agents and ImageProcessing agents (including clipping agents and fire detection agents).

<b>Test Case Id</b>	<b>MT.EOComp.ContractNegotiationMCP</b>	
<b>Summary</b>	Selecting the fastest product for fire detection and negotiating the price.	
<b>Actors</b>	EORequestor-Crisis, ImageProviders, clipping actors, fire detection actors, the ARGUGRID platform	
<b>Preconditions</b>	User preferences will match a crisis situation, in which faster time deliveries will be preferred to other considerations.	
<b>Triggers</b>	EORequestor-Crisis initiates session with the ARGUGRID platform.	
<b>Steps</b>	<b>Step</b>	<b>Result</b>
	1. – EORequestor-Crisis asks for a valid fire detection product, specifying that this needs to be available as soon as possible.	KDE accepts the EORequestor-Crisis request, in the form of an abstract workflow consisting of two abstract services, an image and a fire detector
	2. ArguBroker translate request into a MARGO goal.	A MARGO goal
	3. The MARGO agent receiving the goal (indicated below as EORequestor-Crisis agent) uses GOLEM+PLATON to retrieve suitable service providers with corresponding agents	A number of providers (each given by a service EPR+agent name) all providing the kind of image requested or the fire detection service requested. These providers are “held” within the mind of the EORequestor-Crisis agent
	4. MARGO is used by the EORequestor-Crisis agent to select the provider and the fire detector with the combined fastest delivery time.	Two providers (indicated below as ImProv agent and FireDet agent) “held” within the mind of the EORequestor-Crisis agent
	MARGO realises that it will need a clipping service, as the image from ImProv is not of suitable size for FireDet	New MARGO goals
	EORequestor-Crisis agent uses GOLEM+PLATON to retrieve suitable clipping service providers with corresponding agents	A number of providers (each given by a service EPR+agent name) all providing clipping. These providers are “held” within the mind of the EORequestor-Crisis agent
	MARGO is used by the EORequestor-Crisis agent to select a suitable clipping service to be combined with the services provided by ImProv and FireDet.	A single provider (indicated below as Clip agent) “held” within the mind of the EORequestor-Crisis agent
	5. The ImProv agent consults ArguBroker about SLA templates for the three services to	SLA templates for the image service by ImProv, the clipping service by Clip,

	be provided to EORequestor-Crisis agent. The KDE fetches the SLA templates from GRIA and returns it to the ImProv agent.	and the fire detector by FireDet.
	5. The EORequestor-Crisis agent negotiates with the ImProv agent on price. The EORequestor-Crisis agent starts from the lower-end of the interval offered by the ImProv agent minus some amount. The ImProv agent starts from the upper-end of its allowed interval for price. The agents use the MCP (using their SIM+SDMM). MARGO and GOLEM are involved.	A price for the image.
	The EORequestor-Crisis agent negotiates with the Clip agent on price. The EORequestor-Crisis agent starts from the lower-end of the interval offered by the Clip agent minus some amount. The Clip agent starts from the upper-end of its allowed interval for price. The agents use the MCP (using their SIM+SDMM). MARGO and GOLEM are involved.	A price for the clipping service.
	The EORequestor-Crisis agent negotiates with the FireDet agent on price. The EORequestor-Crisis agent starts from the lower-end of the interval offered by the FireDet agent minus some amount. The FireDet agent starts from the upper-end of its allowed interval for price. The agents use the MCP (using their SIM+SDMM). MARGO and GOLEM are involved.	A price for the fire detector service
	6. The EORequestor-Crisis agent returns the EPR of the service represented by the ImProv agent, its details (delivery time etc) as well as the negotiated price to the ArguBroker. It also returns the EPRs and details of the clipping service and fire detector, with details and negotiated price.	Service descriptions+prices (for the three services) for ArguBroker's consumption
	7. KDE returns to user proposed image provider, clipping, fire detector, their details, and the negotiated prices.	Visualized best option characteristics and price.
	8. – EORequestor-Crisis accepts proposed set of image, services and prices.	KDE orders image acquisition, clipping and fire detector.
<b>Postconditions</b>	Fittest image acquisition and fire detector are shown to EORequestor-Crisis at a negotiated price. One additional service (clipping) is also shown	
<b>Evaluation criteria</b>	<b>Market: B2B/GMV</b> <b>QoR for DM:</b> The chosen combination of services is the same that would have been chosen by GMV. <b>QoR for NC:</b> The price is better than the one obtained by GMV.	
	<b>Market: B2B/GMV and InforSense</b> <b>AL for DM:</b> How long is taken to select the best service providers, with respect to InforSense techniques prior to ARGUGRID? <b>AL for CN:</b> How long is taken to generate the contract (price) automatically? How does this compare to the time taken by business experts (GMV)?	
<b>Notes</b>		
<b>Author, date &amp; version</b>	Francesca Toni, Stefano Bromuri, Visara Bromuri, José Barba, 2008/11/20, v2	

Note that, in this test, the KDE user asks for two single services (an image plus a fire detection processing) but is returned with a combination of three services (an image, a clipping service plus a fire detection processing). The agent is helping the user to solve the problem by refining the original abstract workflow. Each service is provided by the fastest providers among all the valid and available service providers. All the prices are negotiated using the minimal concession strategy of WP4.

### Test Case MT.EOComp.WfContractNegotiationMCPR

Use Case EOComp, described in document D.1.2, and summarized in section 3.1.4 will be used.

In this second Test Case, the EORequestor-Research actor is interested in final products with higher quality or closer fit to its preferences. Price is a strong point in its decision and cheap image providers and image processors will be preferred to expensive ones.

The expected output for this Test Case is the selection of a final “fire detection” product with optimal relation between price and quality, according with EORequestor-Research preferences, if one exists, or the best quasi-match otherwise.

This test is analogous to Test Case MT.EOSelect.WfContractNegotiationMCPR.

<b>Test Case Id</b>	<b>MT.EOComp.WfContractNegotiationMCPR</b>	
<b>Summary</b>	Selecting the highest-quality/best product for a fire detection and negotiating quality and price	
<b>Actors</b>	EORequestor-Research, ImageProviders, clipping actors, fire detection actors, the ARGUGRID platform	
<b>Preconditions</b>	User preferences amounts to best resolution/price deal, given certain constraints on quality and price.	
<b>Triggers</b>	EORequestor- Research initiates session with the ARGUGRID platform.	
<b>Steps</b>	<b>Step</b>	<b>Result</b>
	1. – EORequestor- Research asks for a valid fire detection product, specifying quality boundaries and a price interval.	KDE accepts the EORequestor-Research request, in the form of an abstract workflow consisting of two abstract services, an image and a fire detector
	2. ArguBroker translate request into a MARGO goal.	A MARGO goal
	3. The MARGO agent receiving the goal (indicated below as EORequestor-Research agent) uses GOLEM+PLATON to retrieve suitable service providers with corresponding agents	A number of providers (each given by a service EPR+agent name) all providing the kind of image requested or the fire detection service requested. These providers are “held” within the mind of the EORequestor-Research agent
	4. MARGO is used by the EORequestor-Research agent to select the provider and the fire detector with the closest price to the requested one.	Two providers (indicated below as ImProv agent and FireDet agent) “held” within the mind of the EORequestor-Research agent
	5. MARGO realises that it will need a clipping service, as the image from ImProv is not of suitable size for FireDet	New MARGO goals
6. EORequestor-Research agent uses GOLEM+PLATON to retrieve suitable clipping service providers with corresponding agents	A number of providers (each given by a service EPR+agent name) all providing clipping. These providers are “held”	

		within the mind of the EORquestor-Research agent
	7. MARGO is used by the EORquestor-Research agent to select a suitable clipping service to be combined with the services provided by ImProv and FireDet.	A single provider (indicated below as Clip agent) “held” within the mind of the EORquestor-Research agent
	8. The ImProv agent consults ArguBroker about SLA templates for the three services to be provided to EORquestor-Research agent. The KDE fetches the SLA templates from GRIA and returns it to the ImProv agent.	SLA templates for the image service by ImProv, the clipping service by Clip, and the fire detector by FireDet.
	9. The EORquestor-Research agent negotiates with the ImProv agent on price, using the MCP (using their SIM+SDMM). MARGO and GOLEM are involved. This fails	Failure.
	10. The ImProv agent starts negotiation with the MCP. Agents again use their SIM+SDMM and MARGO and GOLEM are involved.	An image, of price within the given boundaries but slightly lower quality.
	11. The EORquestor-Research agent negotiates with the Clip agent on price. The EORquestor-Research agent starts from the lower-end of the interval offered by the Clip agent minus some amount. The Clip agent starts from the upper-end of its allowed interval for price. The agents use the MCP (using their SIM+SDMM). MARGO and GOLEM are involved.	A price for the clipping service.
	12. The EORquestor-Research agent negotiates with the FireDet agent on price. The EORquestor-Research agent starts from the lower-end of the interval offered by the FireDet agent minus some amount. The FireDet agent starts from the upper-end of its allowed interval for price. The agents use the MCP (using their SIM+SDMM). MARGO and GOLEM are involved.	A price for the fire detector service
	13. The EORquestor-Research agent returns the EPR of the service represented by the ImProv agent, its details (delivery time etc) as well as the negotiated price to the ArguBroker. It also returns the EPRs and details of the clipping service and fire detector, with details and negotiated price.	Service descriptions+prices (for the three services) for ArguBroker’s consumption
	14. KDE returns to user proposed image provider, clipping, fire detector, their details, and the negotiated prices.	Visualized best option characteristics and price.
	15. EORquestor-Research accepts proposed set of image, services and prices.	KDE orders image acquisition, clipping and fire detector.
<b>Postconditions</b>	Fittest image acquisition and fire detector are shown to EORquestor-Research at a negotiated price. One additional service (clipping) is also shown	
<b>Evaluation criteria</b>	<b>Market: B2B/GMV</b> <b>QoR for DM:</b> The chosen combination of services is the same that would have been chosen by GMV. <b>QoR for NC:</b> The price is better than the one obtained by GMV.	
	<b>Market: B2B/GMV and InforSense</b> <b>AL for DM:</b> How long is taken to select the best service providers, with respect to InforSense techniques prior to ARGUGRID? <b>AL for NC:</b> How long is taken to generate the contract (price) automatically? How does this compare to the time taken by business experts (GMV)?	
<b>Notes</b>		

<b>Author, date &amp; version</b>	Francesca Toni, Stefano Bromuri, Visara Bromuri, José Barba, 2008/11/20, v2
-----------------------------------	---

### 5.3 “Factory” tests: Component Validation

Following the bottom-up approach, in a first step, the different components of the ARGUGRID platform will be evaluated.

#### 5.3.1 GRIA component

The GRIA Grid middleware is a service-oriented infrastructure designed to support collaborations through service provision across organisational boundaries. As a part of the ARGUGRID platform, GRIA provides the Grid middleware, where service providers publish their services, are discovered and executed. Furthermore, it provides the Service Level Agreement (SLA) of the service, by means of its Service Provider Management module.

The GRIA software was developed within the framework of the EC IST GRIA project (2001 – 2004) and now is managed by the IT Innovation. Since it is a system developed outside the ARGUGRID project, we assume it has been tested thoroughly as a stand-alone application. However, we want also to test it in order to assess its features to be used within our project. The functionalities under consideration are ease of usage, deployment of services as well as implementation of GRIA services, set up of SLA templates and management of resources.

Therefore, a number of Test Cases were defined in order to evaluate the above features.

#### Test Case FT\_C.GRIA.ExecuteGRIAServices

The basic packages of the GRIA software will be installed in different machines and several complete examples will be run. A user would choose a service to use (the Data and/or the Job service or newly implemented GRIA Services), opening a Trade Account with the service provider and agreeing on an SLA template. He then would execute the service and would monitor its usage. Moreover, we want to test how the SLA module behaves and is linked with the services, since it plays an important role in the ARGUGRID use cases. For that reason, several SLA templates were created, linked to the services and agreed by the users.

<b>Test Case Id</b>	<b>FT_C.GRIA.ExecuteGRIAServices</b>	
<b>Summary</b>	Find a GRIA service, with a valid SLA, and executes the service.	
<b>Actors</b>	ImageProvider; ImageProcessors; Administrator user.	
<b>Preconditions</b>	Installation of the GRIA Grid Middleware’s Client, Service Provider & Basic Applications Packages.	
<b>Triggers</b>		
<b>Steps</b>	<b>Step</b>	<b>Result</b>
	1. – Implementation and deployment of GRIA services. GMV being an expert in the area of Earth Observation (EO) observation has provided different services, such as EO catalogues, clipping, format converters and fire detection services. These have to be deployed as GRIA services creating a web service client to call the GMV service and also using wrapper scripts.	Deployment successful.
	2. – Creation up of SLA templates for each GRIA service. Each service has to be linked with one or more SLA templates. Moreover, different constraints have to be applied to each service as well as a billing element	Valid SLA templates and link with Services.

	that will represent the cost for executing the service.	
	3 – Execution of GRIA services. Although, the execution of the services will be eventually performed via the KDE interface, for testing purposes the GRIA client package will be used for calling and executing the service. The client will operate in a different machine, other than the one with the services, in order to have different administrative domains.	Execution OK.
<b>Postconditions</b>	A GRIA service located, its SLA accepted as valid, and executed.	
<b>Evaluation criteria</b>	Not required for a “factory” test. Only postcondition validation is expected.	
<b>Notes</b>		
<b>Author, date &amp; version</b>	Stella Kafetzoglou, 2008/05/24, v1	

### Test Case FT\_C.GRIA.Registry

Another important component of the ARGUGRID platform is the registries. We have decided to use the GRIA Grid registries in order to store the services, the associated SLA templates as well as XML description files to be used by the peer-to-peer PLATON component. In order to assess its performance and how retrieval of registry objects behaves, different test will be run to ensure perform search and object retrieval.

<b>Test Case Id</b>	<b>FT_C.GRIA.Registry</b>	
<b>Summary</b>	Set up the GRIA registry, store SLA templates and XML files and link them with GRIA services	
<b>Actors</b>	ImageProcessors; Administrator user.	
<b>Preconditions</b>	i) Installation of the GRIA Registry, ii) Loading of SLA templates and XML files iii) Link with the GRIA services	
<b>Steps</b>	<b>Step</b>	<b>Result</b>
	1. – Load the SLA templates and the XML files into the registry. Previously the SLA templates have been created according to the GRIA specifications. The XML files of EPR, the description file of each GRIA Service and the SLA templates are registered into the GRIA Registry of each node under the appropriate concepts. Afterwards they are linked with each other.	Loading successful
	2. – Forming of search queries. Queries are submitted to the GRIA registry in order to retrieve for example the XML representation of the EPR of a service or the respective SLA template. A specific language and query translator were used for the formation of these queries.	Creation of valid search queries
	3. – Search GRIA services based on its characteristics described in the SLA template. The desired characteristics of a GRIA service already registered into the Registry are given as input to the search query. The search query is executed and the GRIA service that best matches with the desired characteristics, according to the SLA that is linked to it, is discovered.	GRIA service returned as requested

<b>Postconditions</b>	Registry has the service descriptions stored internally, available for querying. GRIA services and the respective SLAs are registered into the Registry. A valid search query is submitted to the Registry, the requested GRIA Service that fulfils the desired QoR characteristics is returned. User retrieves a service that matches their description.
<b>Evaluation criteria</b>	Not required for a “factory” test. Only validation is expected.
<b>Notes</b>	
<b>Author, date &amp; version</b>	StellaKafetzoglou, Dimitra Kollia 2008/05/24, v1

### 5.3.2 PLATON component

PLATON is the Peer-to-Peer platform supporting multi-attribute and range queries. It plays a key role in the overall ARGUGRID platform, being responsible for discovering remote agents or Grid services, based on a set of criteria, as defined by an ARGUGRID scenario.

PLATON needs to be validated in terms of its scalability in a large-scale network scenario. To accomplish this, a large number of computers running the PLATON middleware need to be involved. Within simulations, we have achieved to validate proof of correctness and performance efficiency using network scenarios of around 150.000 peer nodes. For a real-world validation, to employ a large number of nodes, we plan to use PlanetLab, a global emulation test bed, offering access to more than 800 computers worldwide.

The following functionality of the PLATON middleware needs to be tested and validated:

#### Test Case FT\_C.PLATON.Routing

In this test, performance of PLATON’s routing protocol in terms of network messages (i.e. hops between peer nodes) would be tested. PLATON needs to resolve multi-attribute queries in a network scenario consisting of N peer nodes.

<b>Test Case Id</b>	<b>FT_C.PLATON.Routing</b>	
<b>Summary</b>	PLATON resolves a request for a 2-dimensional query.	
<b>Actors</b>	No actors involved; Used PLATON peer vs. PLATON peer node.	
<b>Preconditions</b>	<ol style="list-style-type: none"> <li>1. The IP address of a boot-strapping node.</li> <li>2. All 2-D dimensional points representing resources need to be configured in advance within the bootstrapping node.</li> </ol>	
<b>Triggers</b>	A user looking for a particular 2-dimensional point (x,y)	
<b>Steps</b>	<b>Step</b>	<b>Result</b>
	1. – Load all resources (i.e. 2-dimensional points) within the boot-strapping node	“Resources loaded” message.
	2. – Run the PLATON middleware in N different computer nodes.	“PLATON node is up” message, received by all N nodes.
	3. – Issue a 2-dimensional query for a resource that exists in the overall system.	If the resource is found, report the value of the resource and the number of hops taken to resolve the query. Otherwise, a “resource not found” message.
<b>Postconditions</b>	Resources found with expected	
<b>Evaluation criteria</b>	Not required for a “factory” test. Only postcondition validation is expected.	
<b>Notes</b>	This test aims at validating the performance of PLATON’s routing engine in a network of N nodes. Possible values of N are taken from the set {8,16,32,64,128}.	
<b>Author, date &amp; version</b>	Leonidas Lymberopoulos, 2008/05/23, v1	

### Test Case FT\_C.PLATON.LoadBalancing

Performance of PLATON's load balancing algorithm in terms of correctness (i.e. if the final load distribution is the expected) and performance in terms of number of network messages needed for performing the actions required by the load-balancing algorithm.

<b>Test Case Id</b>	<b>FT_C.PLATON.LoadBalancing</b>	
<b>Summary</b>	PLATON re-distributes a given load change when a peer adds new resources.	
<b>Actors</b>	No actors involved; Used PLATON peer vs. PLATON peer node.	
<b>Preconditions</b>	The IP address of a bootstrapping node where N PLATON nodes are already running.	
<b>Triggers</b>	A user adding to the middleware <i>P</i> new 2-dimensional points (x,y) in a network of <i>N</i> nodes.	
<b>Steps</b>	<b>Step</b>	<b>Result</b>
	1. – Load all resources (i.e. 2-dimensional points) within the boot-strapping node.	“Resources loaded” message.
	2. – Run the PLATON middleware in N different computer nodes.	“PLATON node is up” message, received by all N nodes.
	3. – Add <i>P</i> new resources to a randomly chosen peer node.	The result will be the number of messages needed to perform load balancing of the new <i>P</i> points.
<b>Postconditions</b>	Load-balanced node.	
<b>Evaluation criteria</b>	Not required for a “factory” test. Only postcondition validation is expected.	
<b>Notes</b>	This test aims at validating the performance of PLATON's load balancing algorithm in a network of N nodes. Possible values of N are {8, 16, 32, 64, 128}. The test will be performed with different values for the number of new resources <i>P</i> .	
<b>Author, date &amp; version</b>	George Tsatsanifos, 2008/05/23, v1	

### 5.3.3 KDE component

The following are the key functions of InforSense KDE within ARGUGRID:

1. Semantic Workflow Authoring tool allowing service requestors to annotate their abstract workflows with semantic descriptions of their requirements.
2. ArguBroker Component providing a translation interface between KDE and the GOLEM agents. The ArguBroker has two key functionalities. The first is translating from abstract workflows (DPML and WSMO) to agent-understandable descriptions in Prolog. The second is translating from agent-understandable descriptions in Prolog to executable workflows.
3. Semantic Registry Environment allowing service providers to publish semantic information on their services, and allowing service requestors to search and retrieve such descriptions
4. Workflow execution engine accessing and invoking GRIA services.

### Test Case FT\_C.KDE.Authoring

This Test Case allows users (service requestors) to author an abstract workflow and annotate it with semantic service descriptions in WSMO. To achieve this, the KDE GUI is extended with facilities to enable users to enter the WSMO descriptions per abstract workflow node. The annotations are then saved within the DPML file.

<b>Test Case Id</b>	<b>FT_C.KDE.Authoring</b>
<b>Summary</b>	Authoring, annotating and saving abstract workflows.

<b>Actors</b>	EOrequestor-Crisis, EOrequestor-Research, EOrequestor-Private	
<b>Preconditions</b>	Installation of the InforSense Workflow System (Server and Client) and ARGUGRID plug-in	
<b>Triggers</b>		
<b>Steps</b>	<b>Step</b>	<b>Result</b>
	1. – Drag on abstract components and connect them together to construct a workflow. This workflow should not be executable until annotations are added to each of the components	Workflow composed successfully
	2. – Copy a WSMO description from a WSMO editor and paste it into the Notes tab in the Node Editor panel for each abstract component. This description should be parsed and displayed in the WSMO tab showing only the parameters and their value for easier reading. The workflow should now be executable	Annotations added and parsed successfully. Error messages displayed when required
	3. – The abstract workflow should be saved to the Userspace with the annotations included. When the workflow is loaded the annotations should still be attached to the nodes.	Workflow loaded correctly with annotations intact
	4. – Test submission to ArguBroker. When an abstract workflow is complete it should be executable. Once executed a web service call with the DPML of the workflow is expected to be submitted and a concrete workflow returned.	DPML for abstract workflow submitted successfully and concrete workflow is returned.
<b>Postconditions</b>	A concrete workflow which matches the descriptions and conditions specified in the WSMO should be returned.	
<b>Evaluation criteria</b>	Not required for a “factory” test. Only validation (checking the postcondition) is expected.	
<b>Notes</b>		
<b>Author, date &amp; version</b>	Nabeel Azam, 2008/05/24, v1	

### Test Case FT\_C.KDE.Argubroker.DPML\_to\_Prolog

This Test Case tests the ArguBroker DPMLtoProlog functionality ensuring that both the input DPML and output Prolog are legal syntax. The ArguBroker itself is made available through a web service interface accessible from the KDE user interface.

<b>Test Case Id</b>	<b>FT_C.KDE.Argubroker.DPML_to_Prolog</b>	
<b>Summary</b>	Translation from InforSense DPML workflow in XML format to Prolog format	
<b>Actors</b>	EOrequestor-Crisis, EOrequestor-Research, EOrequestor-Private	
<b>Preconditions</b>	Installation of the InforSense Workflow System (Client). InforSense KDE Argubroker deployed as web services.	
<b>Triggers</b>		
<b>Steps</b>	<b>Step</b>	<b>Result</b>
	1. – Create DPML workflow using InforSense KDE system together with WSMO description.	DPML Workflow composed successfully with WSMO embedded.
	2. – Extract the created DPML XML file from InforSense KDE project.	DPML XML file saved.
	3. – Run DPML to Prolog translator using the DPML XML file extracted from the previous step as input.	Translated workflow in Prolog format is displayed on the testing console. It is also saved as workflow.pl file if

		required.
<b>Postconditions</b>	Translated workflow in Prolog format is displayed on the testing console. It is also saved as workflow.pl file if required.	
<b>Evaluation criteria</b>	Not required for a “factory” test. Only validation (checking the postcondition) is expected.	
<b>Notes</b>		
<b>Author, date &amp; version</b>	Li Guo, 2008/05/24, v1	

### Test Case FT\_C.KDE.Argubroker.WSMO\_to\_Prolog

This Test Case tests the ArguBroker WSMOtoProlog functionality ensuring that both the input DPML and output Prolog are legal syntax. The ArguBroker itself is made available through a web service interface accessible from the GOLEM agent.

<b>Test Case Id</b>	<b>FT_C.KDE.Argubroker.WSMO_to_Prolog</b>	
<b>Summary</b>	Translation from WSMO in XML format to Prolog format	
<b>Actors</b>	EOrequestor-Crisis, EOrequestor-Research, EOrequestor-Private	
<b>Preconditions</b>	Installation of the InforSense Workflow System (Client). InforSense KDE Argubroker deployed as web services.	
<b>Triggers</b>		
<b>Steps</b>	<b>Step</b>	<b>Result</b>
	1. – Create WSMO goals using WSMO studio	WSMO goal created
	2. – Export WSMO goal to XML file from WSMO studio	WSMO XML file saved.
	3. – Run WSMO to Prolog translator using the exported WSMO XML file from the previous step as input.	Translated WSMO goal in Prolog format are displayed on the testing console. It is also saved as goal.txt file if required.
<b>Postconditions</b>	Translated WSMO goal in Prolog format are displayed on the testing console. It is also saved as workflow.pl file if required.	
<b>Evaluation criteria</b>	Not required for a “factory” test. Only validation (checking the postcondition) is expected.	
<b>Notes</b>		
<b>Author, date &amp; version</b>	Li Guo, 2008/05/24, v1	

### Test Case FT\_C.KDE.Argubroker.WSML\_to\_Prolog

This Test Case tests the ArguBroker PrologtoDPML functionality ensuring that both the input DPML and output Prolog are legal syntax.

<b>Test Case Id</b>	<b>FT_C.KDE.Argubroker.WSML_to_Prolog</b>	
<b>Summary</b>	Translation from WSML in XML format to Prolog format	
<b>Actors</b>	EOrequestor-Crisis, EOrequestor-Research, EOrequestor-Private	
<b>Preconditions</b>	Installation of the InforSense Workflow System (Client). InforSense KDE Argubroker deployed as web services.	
<b>Triggers</b>		
<b>Steps</b>	<b>Step</b>	<b>Result</b>
	1. – Create WSML services using WSMO studio	WSML services created

	2. – Export WSML services to XML file from WSMO studio	WSML XML file saved.
	3. – Run WSML to Prolog translator using the exported WSML XML file from the previous step as input.	Translated WSML service in Prolog format is displayed on the testing console. It is also saved as goal.txt file if required.
<b>Postconditions</b>	Translated WSML service in Prolog format is displayed on the testing console. It is also saved as workflow.pl file if required.	
<b>Evaluation criteria</b>	Not required for a “factory” test. Only validation (checking the postcondition) is expected.	
<b>Notes</b>		
<b>Author, date &amp; version</b>	Li Guo, 2008/05/24, v1	

### Test Case FT\_C.KDE.Argubroker.Prolog\_to\_DPML

<b>Test Case Id</b>	<b>FT_C.KDE.Argubroker.Translator.Prolog_to_DPML</b>	
<b>Summary</b>	Translation from concrete DPML workflow in Prolog format to executable DPML workflow in XML format.	
<b>Actors</b>	EOrequestor-Crisis, EOrequestor-Research, EOrequestor-Private	
<b>Preconditions</b>	Installation of the InforSense Workflow System (Server and Client). InforSense KDE Argubroker deployed as web services.	
<b>Triggers</b>		
<b>Steps</b>	<b>Step</b>	<b>Result</b>
	1. – Create DPML workflows in Prolog formats by using any text editors.	Concrete DPML workflow files in Prologs created.
	2 – Run Prolog to DPML translator using the created concrete Prolog workflow files from the previous step as input.	Translated executable DPML.dmml file is saved.
	3. – Import the saved executable DPML.dmml file into InforSense KDE client and check it is both syntax and semantic sound.	DPML.dmml file is displayed and checked by InforSense KDE client.
<b>Postconditions</b>	Translated executable DPML.dmml file is saved.	
<b>Evaluation criteria</b>	Not required for a “factory” test. Only validation (checking the postcondition) is expected.	
<b>Notes</b>		
<b>Author, date &amp; version</b>	Li Guo, 2008/05/24, v1	

### Test Case FT\_C.KDE.Registry

This Test Case tests the Semantic Registry Publishing functionality, the user inputs or uploads, the semantic descriptions for the functional and non-functional property for each service, and the registry checks for legal syntax and stores the annotations.

<b>Test Case Id</b>	<b>FT_C.KDE.Registry</b>
<b>Summary</b>	Publish and retrieve semantic service descriptions.
<b>Actors</b>	EOrequestor-Crisis, EOrequestor-Research, EOrequestor-Private
<b>Preconditions</b>	InforSense KDE Argubroker deployed as web services. GMV Oil Spill detection services as GRIA Services

<b>Triggers</b>		
<b>Steps</b>	<b>Step</b>	<b>Result</b>
	1. – Run the client for the Semantic Registry. A web service call should be made to the registry. It should display a list of services already published and a list of service providers.	List of entries in registries loaded successfully on start-up
	2. – A service provider is required for each service. If a service provider entry does not exist a new one must be created. Click the Add Provider button on the client to launch the GUI. Here the details for the service provider can be entered. A web service call is made to the registry to update the entry.	New service provider added successfully.
	3. – Add a new Service by clicking the “Add Service” button. This should pop up a window allowing the details for a service (ID, entry point, description, etc) to be entered. If any values are missing an error dialog box will be displayed. Once successfully completed the service should be added to the registry via a web service call and the details retrievable.	New service added and registry updated as expected.
	4. – Selecting a service allows the end user to view the annotations. If no annotations are available a blank text area should appear. Adding or updating an annotation should launch a new window (see next step). An option to delete the existing annotation should also be present. Deleting the information should make a web service call to the update the service entry.	All buttons appear and function correctly.
	5. – The Add Annotation dialog should allow end users to copy a WSMO description created in a WSMO editor to the service entry. Accepting the annotation should make a web service call to the registry to update the entry.	Annotation added successfully.
<b>Postconditions</b>	Registry updated as required	
<b>Evaluation criteria</b>	Not required for a “factory” test. Only validation (checking the postcondition) is expected.	
<b>Notes</b>		
<b>Author, date &amp; version</b>	Li Guo, 2008/05/24, v1	

### Test Case FT\_C.KDE.Execution.1

This Test Case tests the validation and setting up of a concrete workflow returned by ArguBroker for execution. The system presents to the user the concrete workflow, and for each component displays the selected GRIA service and the associated SLA returned by the agent. The user accepts or rejects the workflow and/or individual SLAs. The upload and download components should also be returned attached to their corresponding services.

<b>Test Case Id</b>	<b>FT_C.KDE.Execution.1</b>	
<b>Summary</b>	Set-up of concrete workflow for execution.	
<b>Actors</b>	EOrequestor-Crisis, EOrequestor-Research, EOrequestor-Private	
<b>Preconditions</b>	Installation of the InforSense Workflow System (Server and Client), GRIA plug-in and ARGUGRID plug-in	
<b>Triggers</b>		
<b>Steps</b>	<b>Step</b>	<b>Result</b>
	1. – A concrete workflow should be returned in DPML format and visualized in the client. It should consist of	Workflow returned and the DPML parsed successfully

	nodes that make GRIA service calls and associated SLAs that match the requirements specified in the WSMO description.	
	2. – The SLAs should be displayed in the SLA panel and (parsed for easy reading) for each GRIA execution node and set to the pending status by default. An error message should also be thrown. The workflow must not be executable until all SLAs are accepted.	SLAs displayed and parsed for easy visualization. Workflow is not executable.
	3. – Accepting an SLA should make a GRIA call and create an instance of an SLA Conversation. The error message should disappear making that particular node executable Once all SLAs are accepted the entire workflow should be executable from start to finish	Calls to GRIA made and workflow client status does change accordingly
	4. – If the workflow is saved to the Userspace the status of the SLA acceptance should be persisted. This way, SLAs do not have to be accepted every time a saved workflow is loaded from the Userspace.	Workflows are persisted correctly
<b>Postconditions</b>	A concrete workflow which matches the descriptions and conditions specified in the WSMO should be returned.	
<b>Evaluation criteria</b>	Not required for a “factory” test. Only validation (checking the postcondition) is expected.	
<b>Notes</b>		
<b>Author, date &amp; version</b>	Nabeel Azam, 2008/05/24, v1	

### Test Case FT\_C.KDE.Execution.2

This Test Case tests the execution of the concrete workflow to see if it returns results.

<b>Test Case Id</b>	<b>FT_C.KDE.Execution.2</b>	
<b>Summary</b>	Execution of executable workflow.	
<b>Actors</b>	EOrequestor-Crisis, EOrequestor-Research, EOrequestor-Private	
<b>Preconditions</b>	Installation of the InforSense Workflow System (Server and Client), GRIA plug-in and ARGUGRID plug-in. A concrete workflow with SLAs accepted.	
<b>Triggers</b>		
<b>Steps</b>	<b>Step</b>	<b>Result</b>
	1. – Once a concrete workflow is returned and all the SLAs accepted it is executable. Executing the last node should invoke the whole workflow including GRIA services and return results.	Workflow executed successfully
	2 – A link to the file on an ftp site should be returned if everything is executed successfully, otherwise appropriate error messages relating to inability to connect to GRIA service or incorrect SLA usage	Results returned successfully or correct messages returned
<b>Postconditions</b>	Clicking on returned link takes you to the correctly clipped image	
<b>Evaluation criteria</b>	Not required for a “factory” test. Only postcondition validation is expected.	
<b>Notes</b>		
<b>Author, date &amp; version</b>	Nabeel Azam, 2008/05/24, v1	

### 5.3.4 GOLEM component

GOLEM supports the multi-agent infrastructure, and allows the discovery of agents. The following functionality will be tested:

- Agent deployment
- Object deployment
- Interaction of agents with other agents and objects

#### Test Case FT\_C.GOLEM.AgentDeployment

<b>Test Case Id</b>	<b>FT_C.GOLEM.AgentDeployment</b>	
<b>Summary</b>	A new agent is started within a container with all the associated knowledge bases and reasoning abilities, sensors and effectors. Other agents are capable to contact the new agent dynamically without intervention of the human users or even the agent's own actions.	
<b>Actors</b>	No actors involved; Agent vs. other agents	
<b>Preconditions</b>	An agent which needs to participate in the multi-agent system, e.g. within a VO.	
<b>Triggers</b>	A user (provider or requestor) wanting to be represented within the system.	
<b>Steps</b>	<b>Step</b>	<b>Result</b>
	1. The new agent is added to GOLEM 2. Some other agent finds it	
<b>Postconditions</b>	The new agent is contacted by other agents.	
<b>Evaluation criteria</b>	Not required for a "factory" test. Only postcondition validation is expected.	
<b>Notes</b>		
<b>Author, date &amp; version</b>	Kostas Stathis & Stefano Bromuri, 2008/10/01, v1.0	

#### Test Case FT\_C.GOLEM.ObjectDeployment

<b>Test Case Id</b>	<b>FT_C.GOLEM.ObjectDeployment</b>	
<b>Summary</b>	A new object is started within a container. Agents are capable to view and inspect its state and functionality without intervention of the human users.	
<b>Actors</b>	No actors involved; Agent vs. other agents	
<b>Preconditions</b>	An object that needs to be part of the multi-agent system in order to support domain-specific, application functionalities	
<b>Triggers</b>	An application developer wanting to develop functionalities of an application.	
<b>Steps</b>	<b>Step</b>	<b>Result</b>
	1. The new object is added to GOLEM 2. Some agent finds it	
<b>Postconditions</b>	The new object contributes to the overall application functionalities by agents interacting with it and achieving high-level application goals.	
<b>Evaluation criteria</b>	Not required for a "factory" test. Only postcondition validation is expected.	
<b>Notes</b>		
<b>Author, date &amp; version</b>	Kostas Stathis & Stefano Bromuri, 2008/10/01, v1.0	

## Test Case FT\_C.GOLEM.Interactions

<b>Test Case Id</b>	<b>FT_C.GOLEM.Interactions</b>	
<b>Summary</b>	Two or more agents communicate in the agent environment using objects. Agents communicate using an object serving as a protocol validator for their interaction.	
<b>Actors</b>	Agents, objects	
<b>Preconditions</b>	An agent which needs to form a new VO.	
<b>Triggers</b>		
<b>Steps</b>	<b>Step</b>	<b>Result</b>
	1. An agent initiates the interaction, using the object as the communication state. 2. Any action carried out by agents is performed on the validation object.	The object will hold the outcome of the interaction
<b>Postconditions</b>	The communication is successful.	
<b>Evaluation criteria</b>	Not required for a “factory” test. Only postcondition validation is expected.	
<b>Notes</b>		
<b>Author, date &amp; version</b>	Kostas Stathis & Stefano Bromuri, 2008/10/01, v1.0	

## 5.4 “Factory” tests: Integration Validation

Following the component-testing step, the interaction between the different components will be evaluated and validated. This level focuses on ensuring that the subsystems function together as a whole system.

### 5.4.1 PLATON – GRIA integration

The purpose of the validation of the integration between PLATON and GRIA is to test that discovery of any GRIA service can be realized through PLATON, linking in this way, remote GRIA registries that are located in distinct locations. To test this implementation, we plan to use a set of nodes running the GRIA middleware, index their services using PLATON’s indexing mechanisms and test that queries on multi-attribute services can be resolved efficiently.

### Test Case FT\_I.PLATON-GRIA.Discovery

The test is about the functionality of the PLATON middleware to discover GRIA services anywhere in the network.

<b>Test Case Id</b>	<b>FT_I.PLATON-GRIA.Discovery</b>	
<b>Summary</b>	PLATON resolves a 2-dimensional query for a GRIA service.	
<b>Actors</b>	No actors involved; PLATON and GRIA components used.	
<b>Preconditions</b>	1. The IP address of a PLATON boot-strapping node. 2. Running the GRIA middleware in N computer nodes.	
<b>Triggers</b>	A user looking for a particular 2-dimensional GRIA service	
<b>Steps</b>	<b>Step</b>	<b>Result</b>
	1. – Run the PLATON middleware in N different computer nodes, the same nodes where GRIA is running.	“PLATON node is up” message, received by all N nodes.
	2. – Post GRIA services to each locally attached PLATON middleware instance (This is implemented by the PLATON-GRIA interface).	“Post resources ok” message received by all N GRIA nodes.
	3. – Issue a 2-dimensional query for GRIA service that can be anywhere among the participating GRIA nodes.	If the resource is found, report the value of the service found and the location of the GRIA node holding the service. Otherwise, a “GRIA

	service not found” message.
<b>Postconditions</b>	Return GRIA service or “not found” message.
<b>Evaluation criteria</b>	Not required for a “factory” test. Only postcondition validation is expected.
<b>Notes</b>	
<b>Author, date &amp; version</b>	Leonidas Lymberopoulos, 2008/05/23, v1

#### 5.4.2 PLATON – GOLEM integration

The purpose of the validation of the integration between PLATON and GOLEM, in a similar way than section 5.4.1, is to test that discovery of agents can be realized through PLATON.

##### Test Case FT\_I.PLATON-GOLEM.DiscSuccess

This test is about discovery of agents within a distributed agent environment.

<b>Test Case Id</b>	<b>FT_I.PLATON-GOLEM.DiscSuccess</b>	
<b>Summary</b>	GOLEM agents look for partners to form a VO, using the PLATON P2P platform to discover them.	
<b>Actors</b>	No actors involved; PLATON and GOLEM components used.	
<b>Preconditions</b>	Incomplete knowledge about requester/providers of EO web services.	
<b>Triggers</b>	The user requests a satellite service.	
<b>Steps</b>	<b>Step</b>	<b>Result</b>
	1. - An agent queries the PLATON P2P platform.	A set of agents is discovered.
	2. - Agent communicates result to the user.	User receives message about agents found.
<b>Postconditions</b>	Agent found.	
<b>Evaluation criteria</b>	Not required for a “factory” test. Only postcondition validation is expected.	
<b>Notes</b>		
<b>Author, date &amp; version</b>	Stefano Bromuri, 2008/05/24, v1	

##### Test Case FT\_I.PLATON-GOLEM.DiscFailure

Discovery of agents within a distributed agent environment; agent not found in the EO scenarios.

<b>Test Case Id</b>	<b>FT_I.PLATON-GOLEM.DiscFailure</b>	
<b>Summary</b>	GOLEM agents look for partners to form a VO, using the PLATON P2P platform to discover them. The PLATON environment fails to discover an agent and the requester informs the user of the failure.	
<b>Actors</b>	No actors involved; PLATON and GOLEM components used.	
<b>Preconditions</b>	Incomplete knowledge about requester/providers of EO web services.	
<b>Triggers</b>	The user requests a satellite service.	
<b>Steps</b>	<b>Step</b>	<b>Result</b>
	1. - An agent queries the PLATON p2p platform.	No agent discovered.
	2. - Agent communicates result to the user.	User receives message about failure.
<b>Alternative path</b>	<b>Step</b>	<b>Result</b>
	1B. - Agents already know all the partners and no query is needed.	No query is done.
<b>Postconditions</b>	PLATON informs of no agents found	

<b>Evaluation criteria</b>	Not required for a “factory” test. Only postcondition validation is expected.
<b>Notes</b>	
<b>Author, date &amp; version</b>	Stefano Bromuri, 2008/05/24, v1

### 5.4.3 GRIA – KDE integration

The GRIA Grid middleware is the component called for the execution of services, after the concrete workflow has been returned to the KDE system from the GOLEM agents. As such, we would like to test its responsiveness and its communication with the KDE component, for calling the services and setting up and retrieving the SLAs for the use of them.

The Test Cases will be based around a single executable workflow. This workflow consists of three services, which are used to find a scene, order the scene and perform clipping. The first Test Case involves changing the arguments slightly so different instances of a service are returned, i.e. executed on a different GRIA server. Each concrete workflow that is returned should be executable and the services resolvable. With this Test Case each time different services are returned the correct SLA associated with that service should be returned. When all the SLAs are accepted they are to be submitted to the GRIA server and ready for execution. The workflow is then in an executable state and the GRIA services can be invoked as trust has been negotiated.

#### Test Case FT\_I.GRIA-KDE.Connectivity

As mentioned before, different services are deployed in the GRIA nodes at ICCS infrastructure (the actual services will run at the GMV server but will be wrapped as GRIA services). The KDE component should be able to access and call the services in order to execute them. Moreover, the KDE should be able to create a Trade Account with the service provider and agree on an SLA for the specific service. The Test Cases include the above-mentioned procedure for different services.

<b>Test Case Id</b>	<b>FT_I.GRIA-KDE.Connectivity</b>	
<b>Summary</b>	KDE client find GRIA services, with valid SLA templates and executes them.	
<b>Actors</b>	User (KDE client), Service providers.	
<b>Preconditions</b>	<ol style="list-style-type: none"> <li>1. Wrapping the GMV services into GRIA services and deployment in ICCS nodes.</li> <li>2. Setting up associated SLA templates.</li> <li>3. Creation of input files.</li> </ol>	
<b>Triggers</b>		
<b>Steps</b>	<b>Step</b>	<b>Result</b>
	1. Implementation and deployment of GRIA services. GMV has provided different services from the EO field. These have to be deployed as GRIA services creating a web service client to call the GMV service and also using wrapper scripts. Secondly, each service is linked with one or more SLA templates, concerning billing and specifics of the service.	Deployment successful.
	2. In order to use a GRIA service that is linked with SLA, a Trade Account has to be set up, between the user and the Service provider.	Trade Account opened successfully.
	3. Add GRIA service into the workflow and agree on the provided SLA template.	Executable workflow containing GRIA services that their SLA has been agreed upon.
	4. Each GMV service requires some input that the user gives into the KDE workflow engine. The workflow is executed and each service is called with attributes given by the input file.	Successful execution of the workflow.
<b>Postconditions</b>		

<b>Evaluation criteria</b>	Not required for a “factory” test. Only postcondition validation is expected.
<b>Notes</b>	
<b>Author, date &amp; version</b>	StellaKafetzoglou, 2008/05/24, v1

### Test Case FT\_I.GRIA-KDE.SLARetrieval

Another important interaction between the KDE and the GRIA component is the link between their registries. KDE would provide to the GOLEM agents the associated SLA for each service, in order to negotiate on their terms. The SLA templates are stored to the GRIA registry and therefore the KDE will make a call to the registry in order to retrieve the SLA template. In order to test the correctness of the link, different Test Cases will be set up that include, calling the registry for SLA template retrieval with different services ids passing as arguments.

This second Test Case requires the user to not accept all the SLAs. In this scenario, the workflow cannot be executed until they are all selected. Here an abstract workflow can be modified (parameters changed) in order to return a different set of services. This time, when the user agrees on all the SLAs the workflow is executable.

<b>Test Case Id</b>	<b>FT_I.GRIA-KDE.SLARetrieval</b>	
<b>Summary</b>	KDE registry retrieves SLA template associated with GRIA services	
<b>Actors</b>	KDE, GRIA registry, Administrator user;	
<b>Triggers</b>	Agents ask for the SLA template of a GRIA service	
<b>Preconditions</b>	i) Installation of the GRIA Registry, ii) Loading of SLA templates and XML files iii) Link with the GRIA services	
<b>Steps</b>	<b>Step</b>	<b>Result</b>
	1. – Loading of SLA templates and XML files and link with the GRIA services. The XML files of EPR, the description file of each GRIA Service and the SLA templates are registered into the GRIA Registry of each node under the appropriate concepts. Afterwards they are linked with each other.	Loading and linking successful
	2. – Call to the registry with specified GRIA service id. The KDE registry provides as input the URI of the required GRIA service and the URL of the specific application service through a search query to the GRIA Registry. The GRIA registry searches for the SLA template that is linked with this specific application service and returns it to the KDE Registry.	SLA template retrieval successful
<b>Postconditions</b>	A GRIA service located, its SLA accepted as valid, and executed.	
<b>Evaluation criteria</b>	Not required for a “factory” test. Only postcondition validation is expected.	
<b>Notes</b>		
<b>Author, date &amp; version</b>	StellaKafetzoglou, Dimitra Kollia 2008/05/24, v1	

#### 5.4.4 GOLEM – KDE integration

The functions that need to be tested for GOLEM-KDE integration are:

1. **Translations from DPML/WSMO to Prolog.** To test this function, a DPML workflow that has WSMO notations attached to it should be specified.

2. **Translations from Prolog to DPML/WSMO.** To test this function, a Prolog representation of a concrete workflow should be defined.
3. **Submission of workflow to GOLEM.** To test this function, a Prolog representation of an abstract workflow has to be specified.
4. **Generation of executable workflow.** To test this function, a Prolog representation of a concrete workflow should be defined.
5. **Creating partially instantiated workflow,** to amend with multiple communications with the agent environment
6. **Semantic Registry query** by an agent to obtain a Web Service

### Test Case FT\_I.GOLEM-KDE.WorkflowCreation

As mentioned above, the Test Cases are based around a single workflow, which has three abstract nodes, namely, getImage, orderScene and clip. To test the integration between GOLEM and KDE, the first use case is to create the workflow using the KDE interface and submit it to ArguBroker. ArguBroker will translate the workflow (including DPML, WSMO) to Prolog representation and then submit it to GOLEM. With this use case, the translations from DPML and WSMO to Prolog and submission to GOLEM can be tested.

<b>Test Case Id</b>	<b>FT_I.GOLEM-KDE.WorkflowCreation</b>	
<b>Summary</b>	An abstract workflow is created.	
<b>Actors</b>	Administrator user; ImageProviders; ImageProcessors.	
<b>Preconditions</b>		
<b>Triggers</b>		
<b>Steps</b>	<b>Step</b>	<b>Result</b>
	User creates an abstract workflow.	Abstract workflow created.
	KDE submits abstract workflow to GOLEM.	GOLEM receives a translated Prolog workflow.
<b>Postconditions</b>	GOLEM contains a Prolog representation of the abstract workflow.	
<b>Evaluation criteria</b>	Not required for a “factory” test. Only postcondition validation is expected.	
<b>Notes</b>	Translation DPML/WSMO-Prolog is tested. Submission to GOLEM is tested.	
<b>Author, date &amp; version</b>	M. Ghanem 2008/05/19, v1	

### Test Case FT\_I.GOLEM-KDE.ConcreteWorkflow

A user connects to a GOLEM container in order to ask the agents inside it to solve an abstract workflow. After the agents perform the necessary computation the user receives back a concrete workflow.

<b>Test Case Id</b>	<b>FT_I.GOLEM-KDE.ConcreteWorkflow</b>	
<b>Summary</b>	Solving an abstract workflow into a concrete workflow.	
<b>Actors</b>	EORequestor	
<b>Preconditions</b>	N/A	
<b>Triggers</b>	The user requests a satellite service.	
<b>Steps</b>	<b>Step</b>	<b>Result</b>
	1. - Connecting to a GOLEM container.	GOLEM connection successful.
	2. - Proposing an abstract workflow.	GOLEM receives abstract workflow.

	3. - GOLEM creates a concrete workflow and sends it back.	User receives concrete workflow.
<b>Postconditions</b>	User (KDE client) can visualize proposed concrete workflow.	
<b>Evaluation criteria</b>	Not required for a “factory” test. Only postcondition validation is expected.	
<b>Notes</b>		
<b>Author, date &amp; version</b>	Stefano Bromuri, 2008/05/24, v1.0	

### Test Case FT\_I.GOLEM-KDE.PartialWorkflow

As FT\_I.GOLEM-KDE.ConcreteWorkflow, with the difference that the agent cannot find one or more of the WSs required.

<b>Test Case Id</b>	<b>FT_I.GOLEM-KDE.PartialWorkflow</b>	
<b>Summary</b>	Solving a partial workflow.	
<b>Actors</b>	EORequestor	
<b>Preconditions</b>	N/A.	
<b>Triggers</b>	The user requests a satellite service.	
<b>Steps</b>	<b>Step</b>	<b>Result</b>
	1. - Connecting to a GOLEM container.	GOLEM connection successful.
	2. - Proposing an abstract workflow.	GOLEM receives abstract workflow.
	3. - GOLEM creates a partial workflow and sends it back.	User receives partial workflow.
<b>Postconditions</b>	EORequestor (KDE client) can visualize proposed partial workflow.	
<b>Evaluation criteria</b>	Not required for a “factory” test. Only postcondition validation is expected.	
<b>Notes</b>		
<b>Author, date &amp; version</b>	Stefano Bromuri, 2008/05/24, v1.0	

### Test Case FT\_I.GOLEM-KDE.Semantic

An agent requires a WSMO Web Service that matches a WSMO GOAL. As a consequence it queries the Semantic Registry in GOLEM.

<b>Test Case Id</b>	<b>FT_I.GOLEM-KDE.Semantic</b>	
<b>Summary</b>	Agent queries Semantic registry.	
<b>Actors</b>	EORequestor	
<b>Preconditions</b>	N/A	
<b>Triggers</b>	The user requests a satellite service.	
<b>Steps</b>	<b>Step</b>	<b>Result</b>
	1. - The agent queries the Semantic Registry.	Semantic registry receives query.
	2. - Semantic registry performs the matchmaking.	A set of result is found.
	3. - Semantic registry communicates matchmaking results to agent.	Agent receives a set of web services that can be used inside the abstract workflow.
<b>Postconditions</b>	EORequestor (KDE client) can visualize set of web services.	
<b>Evaluation criteria</b>	Not required for a “factory” test. Only postcondition validation is expected.	
<b>Notes</b>		

<b>Author, date &amp; version</b>	Stefano Bromuri, 2008/05/24, v1.0
-----------------------------------	-----------------------------------

#### 5.4.5 GOLEM – MARGO integration

Agents in GOLEM are built with an argumentation-based mind. The VO formation is supported by the reasoning of agents, which is performed by MARGO.

- Execute a protocol between two agents. This tests the LCC protocols, the protocol execution engine, and the protocol GOLEM-SIM interface.
- Execute a negotiation protocol that does not result in an agreement.
- Execute a negotiation protocol that results in an agreement.

#### Test Case FT\_I.GOLEM-MARGO.Service

The second Test Case focuses on the ability to select concrete services. A service requester selects a concrete service.

<b>Test Case Id</b>	<b>FT_I.GOLEM-MARGO.Service</b>	
<b>Summary</b>	Select the best web service with respect to the user's constraints and preferences	
<b>Actors</b>	EORequestor-Crisis	
<b>Preconditions</b>	The MARGO agent representing the EORequestor-Crisis knows the clipping services: -clipping1, with a price of 50 Euros and a delivery time of 1800'; -clipping2, with a price of 200 Euros and a delivery time of 900'; -clipping3, with a price of 300 Euros and a delivery time of 300'.	
<b>Triggers</b>	The user's constraints (a price between 50 and 300 Euros and a delivery time between 300' and 1800') and preferences (a delivery time as short as possible) are given to the GOLEM agent representing the EORequestor-Crisis through the MARGO GUI.	
<b>Steps</b>	<b>Steps</b>	<b>Result</b>
	The selection of the services	The subset of selected services.
<b>Postconditions</b>	The subset of selected web services: i) Fulfil the user's constraints; ii) Are the best ones with respect to the user's preferences.	
<b>Evaluation criteria</b>	Not required for a “factory” test. Only postcondition validation is expected.	
<b>Notes</b>		
<b>Author, date &amp; version</b>	Maxime Morge, 2008/04/26, v1.0	

#### 5.5 “Factory” tests: System Validation

In this final step, the whole system will be validated. For this end-to-end validation the Earth Observation scenario defined in document D.1.2 will be used.

#### Test Case FT\_S.ARGUGRID.EndToEndConsumerScenario

<b>Test Case Id</b>	<b>FT_S.ARGUGRID.EndToEndConsumerScenario</b>
<b>Summary</b>	End to End Scenario
<b>Actors</b>	EORequestor-Crisis, EORequestor-Research, EORequestor-Private, Administrator User
<b>Preconditions</b>	Installation of the InforSense Workflow System (Server and Client) and ARGUGRID plug-in. GRIA Server to connect to. Installation of ArguBroker, Registries and GOLEM.

Triggers		
Steps	Step	Result
	1. – End User creates abstract workflows using abstract components. Once the abstract workflow is complete, WSMO annotations, created using a WSMO editor, are added to the components. The workflow should now be executable. When executed a web service call is made to ArguBroker submitting the DPML as part of the SOAP message.	Workflow created and annotated. DPML submitted successfully on execution
	2. – After receiving the abstract DPML workflow, ArguBroker translates it into Prolog format. The translation contains two parts: translation from DPML in XML to DPML in Prolog and translation from WSMO in XML to WSMO in Prolog. After the translations are done, the whole translated package is send to GOLEM.	DPML and WSMO in XML are translated into Prolog format. The translated package is sent to GOLEM
	3. – The GOLEM agent receiving the workflow identifies the Abstract Web Services that cannot be solved in isolation and discovers agent providers that could fulfil them. The providers, once the query is received, use the MARGO agent mind to decide which is the best Web Service retrieving it from the ArguBroker	A set of agents interacting with the KDE ArguBroker.
	4. – GOLEM contacts ArguBroker to retrieve service information from semantic registry. It submits its query to ArguBroker. ArguBroker translates the Prolog query into RDQL format and then forwards the RDQL query to semantic registry. A list of matched services is returned as results to ArguBroker.	A list of matched services based on the query submitted is fetched from semantic registry.
	5. – ArguBroker contacts GRIA registry to fetch the SLA template for the retrieved services from step 5. It then translates SLA template into Prolog format for each of the services and returned the final results to GOLEM	SLA templates for services are retrieved from GRIA registry and are returned to GOLEM
	6. –The GOLEM agents, after the negotiation of the workflow and related contract, return a concrete workflow in Prolog format to ArguBroker. ArguBroker first translate it into XML format. The translation includes: WSML translation from Prolog to XML, SLA template translation from Prolog to XML and DPML workflow translation from Prolog to XML. After translation, shimming function is performed to make the workflow executable. Executable workflow is returned from ArguBroker to KDE client.	Executable DPML workflow is created and is returned to InforSense KDE client
	7. – The concrete workflow returned is parsed and displayed in the client. All the SLA information is also parsed and appears in the SLA tab for validation. This workflow must not be executable until the SLAs are accepted.	Workflow returned and displayed correctly. All SLA information is presented successfully. Workflow is not executable as expected.
	8. – Accepting an SLA allows that particular service to be executed and also makes a call to the GRIA server to create an SLA Conversation object from the template provided. Once all the SLAs are accepted the workflow is executable.	SLA acceptance successfully transforms concrete workflow to executable. SLA conversations generated
	9. – Executing the workflow will make the appropriate GRIA service calls using the correct SLAs submitted. Once completed, a link to the ftp site containing the result is returned.	Workflow executed successfully with GRIA service calls and correct SLAs used.
Postconditions	The final image returned from the GMV clipping service	

<b>Evaluation criteria</b>	Not required for a “factory” test. Only postcondition validation is expected.
<b>Notes</b>	
<b>Author, date &amp; version</b>	Nabeel Azam, Li Guo, 2008/11/26, v1

## 6 Conclusions

We have defined our testing methodology and provided a concrete list of tests and a concrete testing environment for the experimentation with the ARGUGRID platform, its components and integrations in two of the scenarios identified in WP1 in the first year of the project.

The proposed methodology is based on solid software engineering principles. Test Cases are commonly used for validating software products, and the “Test in the large” approach (component + integration + system) is widely used for testing systems based on integration of different components.

Bottom-up integration allows several teams to work in different parts of the system in an independent way. Components can be tested as soon as they are available. And, as integration progresses, the system and the new functionalities can be tested, until the whole system is finally integrated and validated.

The classification of the Test Cases into “factory” and “magazine” tests facilitates the evaluation of the platform. This evaluation, focussed on the added value functionalities of the ARGUGRID platform, will clearly show the fulfilment of the project’s aims. We have provided clear and quantifiable ARGUGRID-specific criteria to indicate the added value of our solutions. We have also given a precise description of our intended market and a plan to evaluate our solutions within this market.

The experimentation environment is mostly based on real Grid services, built from web services given by GMV. Although simple, these services are realistic and suited to support the scenarios depicted in deliverable D.1.2.

The list and experimentation environment given in this deliverable may undergo change in the third year, subject to further developments in the ARGUGRID platform and scenarios, but they are a very concrete starting point to validate the ARGUGRID approach. We expect that experimentation will provide very fruitful feedback on the platform, its components and interactions amongst these components. We also foresee that experimentation will push the boundaries of scenarios and use-cases, and may identify interesting variants of these allowing demonstrating in full the innovative functionalities of the ARGUGRID approach to Grid computing and Service-Oriented computing.

## Annex I: Test Case Template

<b>Test Case Id</b>	<i>Test identifier as described in section 3.</i>	
<b>Summary</b>	<i>Description of the Test Case.</i>	
<b>Actors</b>	<i>List of actors than participate in Test case.</i>	
<b>Preconditions</b>	<i>List of conditions (if any) needed for the right execution of the Test File.</i>	
<b>Triggers</b>	<i>Condition that activates the Test Case. It can be one of the preconditions, but this is not necessary.</i>	
<b>Steps</b>	<b>Step</b>	<b>Result</b>
	<i>1. – Step 1 description.</i>	<i>Expected results for step 1.</i>
	<i>2. – Step 2 description.</i>	<i>Expected results for step 2.</i>
	<i>3. – Step 3 description.</i>	<i>Expected results for step 3.</i>
	<i>...</i>	<i>...</i>
	<i>N. – Step N description.</i>	<i>Expected results for step N.</i>
<b>Alternative Path (optional)</b>	<b>Step</b>	<b>Result</b>
	<i>Xb. – Alternative step X description.</i>	<i>Expected results for alternative step Xb.</i>
	<i>Xc. – Alternative step X description.</i>	<i>Expected results for alternative step Xc.</i>
	<i>...</i>	
<b>Postconditions</b>	<i>List of conditions expected after finalization of Test Case.</i>	
<b>Evaluation criteria</b>	<i>List of aspects in which the evaluation of this test should focus. This field is not applicable for “factory” tests.</i>	
<b>Notes</b>	<i>Any needed comment.</i>	
<b>Author, date &amp; version</b>	<i>Name, YYYY/MM/DD, vX.X</i>	

## Annex II: Validation and Evaluation Report Template

Validation and Evaluation Report		
<b>Test id:</b>	<i>Identifier of Test Case executed.</i>	
<b>Run n°:</b>	<i>Identifier of Test Case execution number (several executions could be needed if previous executions fail).</i>	
<b>Date:</b>	<i>YYYY/MM/DD</i>	
<b>Environment &amp; configuration information:</b>	<i>Information about software versions, actors used, configuration info...</i>	
<b>Operator</b>	<i>User that executes the Test Case.</i>	
<b>Status</b>	<input type="checkbox"/> Pass	<input type="checkbox"/> SW Error <input type="checkbox"/> Update Test
<b>Comments</b>	<i>Provide any relevant comment.</i>	
<b>Evaluation</b>	<i>Result of the evaluation, based on the criteria defined in the associated Test Case.</i> <i>This field is not applicable for “factory” tests, and validation result will be reported here.</i>	

## Annex III: Bug Report Template

BUG REPORT	
<b>Test Id:</b>	<i>Test Case identifier.</i>
<b>Date:</b>	<i>YYYY/MM/DD</i>
<b>Summary:</b>	<i>Brief description of error found.</i>
<b>Overview Description:</b>	<i>More detailed expansion of summary.</i>
<b>Steps To Reproduce:</b>	<i>The minimal set of steps necessary to trigger the bug. Include any special setup steps.</i>
<b>Actual Results:</b>	<i>What the application did after performing the above steps.</i>
<b>Expected Results:</b>	<i>What the application should have done, if the bug was not present.</i>
<b>Additional Information</b>	<i>Any other debugging information.</i>

## References

[1]Maxime Morge and Paolo Mancarella (eds) Prototype for argumentative agents. ARGUGRID Deliverable Document D2.1 (2007).

[2]SWI-Prolog is a comprehensive Free Software Prolog environment, licensed under the Lesser GNU Public License available at <http://www.swi-prolog.org/>

[3]SICStus Prolog is a commercial ISO standard compliant Prolog development system which can purchase at <http://www.sics.se/isl/sicstuswww/site/index.html>

[4]The electronic tool based on MS-Excel developed by the Office of Government Commerce, (UK) available at [http://www.ogc.gov.uk/documents/eAuction\\_Decision\\_tool.xls](http://www.ogc.gov.uk/documents/eAuction_Decision_tool.xls) (last accessed May 2008).

[5]Thannasis Stournaras (eds) eBusiness application scenarios. ARGUGRID Deliverable Document D1.2 (2007).