Model and Inference Driven Automated testing of Services architectures

Deliverable D2.1

Requirements for automatically testable services and services architectures
## Revision Chart

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Author (Partner)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.3</td>
<td>30/01/2013</td>
<td>DEDA</td>
<td>Template</td>
</tr>
<tr>
<td>0.4</td>
<td>10/02/2013</td>
<td>SEF</td>
<td>Initial draft</td>
</tr>
<tr>
<td>0.5</td>
<td>12/02/2013</td>
<td>SEF, UGOE, FF</td>
<td>Draft with UGOE and FF contributions</td>
</tr>
<tr>
<td>0.6</td>
<td>25/03/2013</td>
<td>SEF, UGOE, FF</td>
<td>Second draft revised by SEF with UGOE and FF new contributions and CNR review</td>
</tr>
<tr>
<td>0.7</td>
<td>21/05/2013</td>
<td>SEF, UGOE, FF, DEDA, ITA,</td>
<td>Third draft revised by UGOE, FF, SEF and contribution by DEDA and ITA (pilots) and SEF (example)</td>
</tr>
<tr>
<td>0.8</td>
<td>06/06/2013</td>
<td>SEF, UGOE</td>
<td>Fourth draft revised by UGOE and SEF. Bibliography merged. Added project presentation, document structure, executive summary. Amended example.</td>
</tr>
<tr>
<td>1.0</td>
<td>20/06/2013</td>
<td>SEF, UGOE</td>
<td>Candidate final version.</td>
</tr>
<tr>
<td>1.1</td>
<td>24/06/2013</td>
<td>SEF, FF</td>
<td>Final version</td>
</tr>
</tbody>
</table>
EXECUTIVE SUMMARY

With the spreading of the digital economy, a growing number of applications, systems and devices are connected and collaborate without human intermediation, allowing the automation of business processes that support daily activities and services¹. The functional and non-functional dependability and security of such distributed architectures become more and more a critical issue. Dependability and security (functional and non-functional) are firstly the outcome of sound engineering practices.

Service Oriented Architecture (SOA)² is a design and implementation style that allows organizations to put in practice dynamic collaborations of autonomous, heterogeneous and loosely coupled digital systems as direct service exchanges, in order to achieve flexible, reliable and secure business process automation. SOA style is currently practiced following approaches that are situated in a range between two extremities: (i) “weak” interface-based SOA and (ii) “strong” contract-based SOA³.

The “weak” interface-based SOA style is based merely on the design of an Application Programming Interface (API) as a primary access device to some system functionalities and on the management of this interface as a “first-class” artifact that is separate from the system internal implementation.

In the contract-based “strong” SOA style, beyond the separation between interface and implementation and the hiding of systems’ internals that are a must in any case, a service is represented by an artifact referred to as a service contract that incorporates the formal definitions not only of the service interface, but also of the service function and the external behaviors of the parties, including security and performance aspects. The service contract acts as an agreement between the parties and as a collection of requirements and a specification for the implementers.

The spread of interface-based SOA is revealed by the impressive growth of the “API economy”\(^4\). APIs are a key growth driver for hundreds of companies across a wide range of industry sectors and are going to be a primary access channel to technology-driven products and services. The spread of contract-based SOA is slower than one of interface-based SOA, but follows the diffusion of the model-driven engineering approach and concerns today a growing number of distributed system architectures in critical business sectors, where is often linked to standardization initiatives\(^5\). In several services architecture, the contract-based style often gets along with the interface-based one.

In a sense, SOA engineering, as all other engineering activities, is always model-based. We always create a mental model of the reality that we have either to design and build (prescriptive model) or to analyze and assess (descriptive model). The model always exists; the only option is about its form: it may be mental - existing only in the head of the designer, analyst ... - or explicit. An explicit model is potentially sharable between humans; an explicit and formal model of a system may be mechanically transformed until automatic generation of parts of the system. This is the objective of the OMG Model Driven Architecture initiative\(^6\), and the goal of a more general trend named MDE (Model Driven Engineering).

The SOA approach is characterized by a sharp separation between the functional (in a broad sense), black-box model and the constructional, white-box model of a system. With respect to SOA, the inescapable limit of the MDA approach is that the white-box model of a system cannot be mechanically derived by its black-box model, even by the most detailed one. The black-box model (service model) acts as an agreement (service contract) between the supplier of the functionality (the service provider) and its user (the service consumer), while the white-box models are private and hidden. This is not an abstract issue: a published API is an agreement between the service provider and the consumers as the only means that allows interacting with the provider to coordinate the service provision/consumption.


\(^5\) Healthcare Services Specification Program (HSSP) - http://hssp.wikispaces.com/

\(^6\) http://www.omg.org/mda/
How can the service consumer improve reasonably her/his confidence in the compliance of the actual service provision with exigencies and constraints stated in the service contract? The only answer is: by testing. Beyond service compliance, how can the service consumer improve reasonably her/his confidence that the provider is not vulnerable to malicious attacks that can jeopardize the resources handled in behalf of the consumer? The answer is still: by testing. Because of implementations are mutually hidden – even if they are not, it might be too complex and expensive to assess each other implementations by the deep analysis of the internals and white-box testing – black-box testing is the only means available to businesses to improve their trust in their partners’ service provisions.

But SOA testing has the paradoxical trait that the same peculiarities that make it necessary make also it hard due to: lack of observability of the involved systems; lack of trust in the employed engineering methods; lack of direct control of the implementation lifecycles; late binding of systems; fundamental uncertainty of the test verdicts; organizational complexity; elastic demand of computational resources; increasing scale factor of the services architectures; high costs and, last but not least, questionable efficacy of humans in performing manually a more and more hard and complex but boring and low rewarding activity such as testing of large distributed architectures.

In effects, hand-writing of test cases, manual configuration of test environments, manual scheduling of test runs and eyeball assessment of test outcomes are not only labor intensive and difficult to put in practice, but are also the least effective solution of the SOA testing problem, that cannot be overcome by mere methodological recommendations on fundamentally human-based engineering practices.

The solution of the SOA testing problem can be brought only by a disruptive innovation that drastically simplifies and routinizes the testing task by implementing and offering an automated, effective, accessible and affordable testing facility.

The goal of the MIDAS project is to design and build a test automation facility that targets SOA implementations. The MIDAS functionalities shall be available on demand, i.e. on self-provisioning, pay-per-use, elastic basis. The MIDAS facility shall be a multi-tenancy SaaS (Software as a Service) deployed on a cloud infrastructure and accessible in the Internet. The MIDAS testing approach is non-intrusive on the SUT
(System Under Test): the SUT is deployed on its environment (on premise and/or on cloud) and the MIDAS facility interacts with it using the services that it publishes.

The targets of the MIDAS facility are both “weak” interface-based and “strong” contract-based SOAs. The MIDAS facility is intended to provide automation of the “core” testing activities: test generation, scheduling, execution and arbitration.

MIDAS test automation is based on models. Like SOA engineering, SOA testing, and, in general, testing is always model-based. We always create a mental model of the system behavior in order to test it. As for engineering, the model always exists; the only option is about its form: it may be mental - existing only in the head of the tester - or explicit. An explicit model is potentially sharable between humans; an explicit and formal model of a system can be mechanically transformed until the automatic generation of test cases and oracles and the automatic configuration of the test running environment.

The underlying idea is that the testing activity shall shift from test case hand-writing, manual configuration of test environments, manual scheduling of test runs and eyeball assessment of test outcomes – all these activities being conducted by professional testers - to model authoring by designers and architects. The burden of test generation, scheduling, execution and arbitration, until the production of the test report, is given up to the MIDAS facility. The involved models are on one side service and system models and on the other side models of the test goals, means and courses of actions. The general idea is that the deep testing knowledge of professional testers and of the research community can be embedded in the implemented automated testing methods.

What kind of testing methods, approaches and practices will be supplied by the MIDAS on demand automated test facility?

The MIDAS project shall put into operation a substantial collection of functional (unit testing, choreography, orchestration and composition testing) and security (fuzz, security policy compliance) testing methods. Moreover, the automated scheduling and even the dynamic automated test generation shall be managed by probabilistic (Bayesian) inference methods. Furthermore, the new promising usage-based testing approach that is a testing meta-method that considers the usage of the system in the field (in operation) a source of interesting data, information and knowledge (Markov models), shall be carried out. The idea is to use this
knowledge, automatically concentrated in the usage profile by intelligent processing of usage data, for driving the strategy and planning of functional test. The MIDAS facility shall support usage observation on the system in the field with facilities that allow the generation and download of the usage observation software and the upload and arrangement of usage journals.

The MIDAS portfolio of automated test methods, practices and approaches is not closed. From the point of view of the test method designers and developers, the MIDAS on demand automated test facility is an open platform. MIDAS shall implement the concept of test scheme, which is an implemented testing method able to perform automated test generation, scheduling, execution and arbitration.

Hence, according to the already canonical SaaS approach, there are two categories of MIDAS “users”: the end user and the test scheme developer. The end user acts automated testing by (i) supplying models, (ii) deploying accessible systems under test (SUTs) and (iii) invoking against the SUTs the appropriate test schemes that s/he offered by the evolving MIDAS portfolio. The test scheme developer designs and builds test schemes in a format that is MIDAS-compatible, and uploads them as plug-ins on the MIDAS facility in a strictly controlled way. Test schemes are organized, built and formatted in such a way that not only they can be integrated in the MIDAS facility, but also reuse preexisting independent resources such as SUT models and test configuration models. The SUT model and the test configuration model that are already available on MIDAS are reusable by the test schemes. Eventually, the end-user shall provide only the specific information – supplied in the form of test scheme related models - needed by the testing method the invoked test scheme implements.

The MIDAS core functionalities for both end users and test scheme developers shall be presented through APIs. This access modality allows seamless integration with Integrated Development Environments (IDEs) and Application Lifecycle Management (ALM) platforms - for instance, the programmed invocation of automated non-regression test campaigns at specific milestones of a software engineering cycle - and, obviously, new engineering service compositions that are unforeseen at that time.

---

7 The compatibility requirements and recommendations for test scheme developers are not in the scope of this document.
Not all services architectures being equally testable, in general and in particular with MIDAS, we are obliged to state a limited collection of requirements and recommendations that respectively must and should be fulfilled in order to employ the MIDAS facility for SOA testing.

MIDAS test compatibility requirements must be fulfilled by any SUT that is sought to be a MIDAS testing target. With respect to MIDAS compatibility requirements, our work has been driven by two methodological principles: (i) minimize the number of requirements and their enactment burden for the user; (ii) formulate only requirements whose fulfillment improves the general dependability, security, interoperability, conformance with standards and, last but not least, auditability and testability as a generally accepted criteria - non-specific to MIDAS - of single-node and multi-node services architectures.

MIDAS compatibility requirements are independent from any test scheme. Conversely, MIDAS compatibility recommendations are classified as general vs. test scheme specific. General recommendations are related to the use of optional features of the MIDAS facility that improve the testing process but are independent from specific test schemes. The use of these features is optional, but the fulfillment of the related recommendation is a prerequisite for the use of them.

Test scheme specific recommendations have to be adopted only if the user wants to invoke the specific test scheme. The invocation of a specific test scheme is optional for the user, but, whether the user wants to invoke it, the fulfillment of the related recommendations will be a prerequisite. Anyway, the MIDAS facility proposes some basic test schemes that will be operational without requesting the satisfaction of any specific recommendation - only the general compatibility requirements must be fulfilled.

All compatibility requirements and recommendations can be classified in two categories: (i) those that bear on models that are needed, in general or related to specific test schemes, to drive the automated test generation, scheduling, execution and arbitration, (ii) those that bear on the SUT configuration and deployment, in order to consent basic and enhanced binding, connection and interaction of the MIDAS facility with the SUT.

General requirements on models concern “architectural” models, i.e. system models and test configuration models. The system model is descriptive model of the MIDAS target services architecture. The test
configuration model is a \textit{prescriptive model} that is used by the model-driven automatic generation of the test execution system.

The system model includes a \textit{service model} and a \textit{SUT model}. The service model is a \textit{class} model, while the SUT model is an \textit{instance} model of a concrete, physically deployed SUT.

According to the MDA approach, we distinguish between the service platform independent model (service PIM) and the service platform specific model (service PSM), which is a model on a specific technical interoperability platform such as SOAP or REST.

The service PIM is a \textit{standard SoaML model}, compliant with the OMG Service oriented architecture Modeling Language (SoaML) Specification. The service PIM is an abstract, disembodied model of the services implemented by the SUT, which uses two kinds of stereotypes: the \textit{Service Contract} and the \textit{Participant}. The \textit{Service Contract} describes the service abstract specification, disembodied from specific provider and consumer systems, whereas the \textit{Participant} describe a class of abstract systems that realizes a number of service roles (described within the \textit{Service Contract}).

The MIDAS facility needs the availability of the \textit{essential service PIM}, which includes only the minimal information that is necessary to characterize and classify the SUT \textit{nodes} and \textit{ports} (and also the test component elements of the test configuration model - see below). A service PIM may include more enhanced information about protocols and choreographies, which have been produced by a model-driven, contract-first SOA engineering cycle and may be used by specific test schemes.

The service PSM is the service implementable model (WSDL, WADL ...) on the SUT interoperability platforms (SOAP, REST ...) that allows the configuration of the needed connections between the MIDAS facility and the SUT. The accuracy and consistency of the service model (PIM and PSM) is crucial.

The SUT model is a model of a concrete system but is platform independent: it is a UML \textit{Deployment} model that describes the \textit{topology} of the SUT and the \textit{locations} of its accessible nodes/ports. The elements of the SUT model are classified by the elements of the service model. The deployed SUT node/ports (and their locations) described by the SUT are made \textit{accessible} by the MIDAS facility by means of this description. Deployed node/ports that are not described by the SUT model are not
accessible by the MIDAS facility. The accuracy of the SUT model is crucial for sound testing practices.

On the basis of the SUT model, the test configuration model (TC model) prescribes the architectural configuration of a test scenario in a manner that is independent by the test scheme that utilizes it. It indicates: (i) the SUT accessible nodes that shall be the targets of stimuli, responses and observations in the course of the test run, (ii) the SUT accessible places - ports and communications paths - where the stimuli, responses and observations shall be acted and (iii) the connections that shall be established between SUT and the MIDAS test execution environment.

According to the MDA approach, we distinguish between the test configuration platform independent model (TC PIM) and the test configuration platform specific model (TC PSM), which is a model on a specific test platform and environment, such as the TTCN-3 platform\(^8\).

The TC PIM is a UML Component model. TC PIM Components are stereotyped as Proxy, Emulator andInterceptor. Proxies prescribe components that represent the SUT node/ports being the targets of stimuli and responses by the MIDAS facility. Emulators prescribe components that emulate SUT nodes and virtual environment nodes – nodes that are not in the composition of the SUT and represent human or artificial actors that interact with the system. Interceptors are able to place themselves virtually on a communication path between two SUTs (in fact between the representative Proxies) and interact with them. Emulators and Interceptors are architectural placeholders whose testing operational semantics is determined by the test schemes that utilize them.

General compatibility requirements on the SUT bear on the interoperability platforms that are compatible with the MIDAS facility and the availability for a SUT of an initialization and recovery procedure that can be invoked by the MIDAS facility. General compatibility recommendations on the SUT suggest the implementation of ancillary services and tools that make easier the SUT deployment, the check of the models’ accuracy and of the connections.

The other compatibility recommendations are related to specific test schemes, in the fields of functional, security and usage-based testing.

---

\(^8\) [http://www.ttcn-3.org/](http://www.ttcn-3.org/)
Functional testing, practiced as unit (single node) and integration (multi-node cooperation) testing, can be enhanced by the availability of service model elements that specify, beyond the service interface, the function (what the provider does in behalf of the consumer) and the behavior (how the parties interact to coordinate the service provision/consumption). Function specifications that are compatible with the MIDAS facility are grounded on contract-based design: a function is specified by the operation signature and its pre/post-conditions\(^9\). Function models allow the automated generation of tests and oracles.

Stateful service providers, i.e. providers that change durably the state of the resources they handle in behalf of their consumers, because of the SOA approach that forbids the direct inquiry of the aforementioned states, can be tested more effectively only by cross-checking. Cross-checking is put in practice by retrieving internal states through basic transparency services based on international standards that should be implemented by the SUT and matching this information with the SUT responses.

Choreography testing can be applied to end-to-end service exchanges on multi-node services architectures. Composition testing enhances choreography testing with the help, for each involved node, of the transfer function\(^10\), i.e. the formal specification for the involved nodes of the correspondence between the input stimuli and the output interactions that are service composition effects of the stimuli.

Compatibility recommendations for functional testing, on models and on the SUT, are cumulative: the availability of more models (protocol, function, choreography and composition) and of more ancillary services on the SUT (state re-initialization, transparency services) allows more and more enhanced testing methods.

Security testing can be classified into two categories: (i) security-policy compliance testing and (ii) vulnerability testing. Security policies are included in the service contract and security-policy compliance testing is close to functional compliance testing. Vulnerability testing aims at

\(^9\) The MIDAS compatible formulation of operation signatures and pre/post-conditions is OCL (OMG Object Constraint Language (OCL) Version 2.3.1. formal/2012-01-01. Object Management Group. URL http://www.omg.org/spec/OCL/2.3.1/).

\(^10\) Also for the transfer function the MIDAS-compatible formulation is OCL.
seeking *faults* and *weaknesses* (the latter being not necessarily faults from the functional point of view) that have a security impact.

In order to look for such vulnerabilities, the testing perspective is moved from the system specifications to the *attacker behavior*. The main aspect of security testing is to stimulate the SUT with inputs that reveal vulnerabilities. Mostly, such inputs are invalid in the sense of the specification. Therefore, in contrast to functional testing, security testing is mostly negative testing and may be based on *misuse cases* instead of *use cases*.

We are going to develop two kinds of fuzzing approaches: *data fuzzing* and *behavioral fuzzing*. With data fuzzing the SUT is exercised with invalid input data while behavioral fuzzing consists of submitting invalid message sequences to the SUT.

In respect to SOAP-based web services, Representational State Transfer Interfaces (REST) and JavaScript Object Notation (JSON) are getting more popularity in the API economy. Data exchange via JSON is, in contrast to XML documents, less restrictive because of the lack of a document format description that prevents the validation of the document against this description. Hence, the general availability of even simple data fuzzing of JSON documents can be relevant and effective for testing purposes.

The compatibility recommendations on models for security testing concern the availability of interaction protocol models, security policy models, encryption/decryption and signature/validation algorithms and keys utilized by the SUT. The compatibility recommendations on the SUT are the same as those for functional testing.

SOA usage-based testing is a new promising research and engineering approach about SOA testing. The first intent is to focus on highly used functions of the service and highly usual end-to-end service exchanges in services architectures. Test cases are generated such that the testing effort focuses on the highly used parts of the SUT. As a by-product of this approach, it is possible to determine the *reliability* (the rate of failures given a usage scenario) of the SUT with respect to the *current usage* and utilize this measure as a criterion within *acceptance testing*. Conversely, focusing on *lowly used paths of interaction* within the SUT is interesting for augmenting the functional test coverage. Usage-based testing is intended to support the functional testing activities of the MIDAS facility.
Usage-based testing is grounded on the concept of usage profiles. The usage profile describes the usage of a system in stochastic terms, such as the probability of the next interaction. Usage profile models are built from information extracted from usage journals that are issued from usage observation on the system in the field (SIF). The MIDAS facility shall support the user in producing easy-to-put-in-place mechanisms (observer software) that are able to perform usage observation with minimum overhead. These software components, which are placed by the SIF administrator on the chosen SIF usage observation points, are able to provide MIDAS-compatible usage journals, which shall be collected and uploaded on the MIDAS facility.

The crucial recommendation is that the usage data observation mechanisms shall engender usage observation data that should be not only compatible with the MIDAS facility but also accurate, which means that they should represent faithfully the actual usage of the SIF. The fulfillment of this recommendation is essential for sound usage profile modeling.

The first recipients of this document are the MIDAS early adopters, such as the MIDAS project partners that are in charge of the pilots (Healthcare Generic Services pilot / Supply Chain Management pilot). This document will constitute the support that allows them to start by trying the fulfillment of the MIDAS compatibility requirements and recommendations on their SUTs and SIFs. Relatively to these points, the feedback from the MIDAS pilots’ experience will be precious all along the project. If some of the MIDAS compatibility requirements and recommendations will change as a consequence of design and implementation choices and of pilots’ feedbacks, updated versions of this document including these changes will be made available.

---

11 This document should not be considered a user guide, but only a requirement/recommendation specification.
Abstract

<table>
<thead>
<tr>
<th>Project Number</th>
<th>318786</th>
<th>Project</th>
<th>MIDAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Title</td>
<td>Model and Inference Driven Automated testing of Services architectures</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Project website</td>
<td><a href="http://www.midas-project.eu">http://www.midas-project.eu</a></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Deliverable**

| Deliverable Number | D2.1 | Title | MIDAS compatibility requirements and recommendations for services and service architecture |

**Work Package**

| Work Package Number | WP2 | Title | SOA testing framework general architecture |

**Task**

| Task Number | T2.1 | Title | Requirements for automatically testable services and services architectures |

**Date of Delivery**

<table>
<thead>
<tr>
<th>Date of Delivery</th>
<th>Contractual</th>
<th>Actual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>draft</td>
<td>final</td>
</tr>
</tbody>
</table>

**Dissemination Level**

| Public | Confidential |

**Summary for dissemination**

This is a report that contains the requirements and recommendations about (i) service and service architecture specifications and implementation and (ii) service architecture lifecycle and (iii) service architecture deployment and operation in order to guarantee automated testability.

**Keywords**

SOA, services architecture, service orientation, testability

**Contact Person**

Libero Maesano (SEF)

**Authors (Partner)**

Fabio De Rosa (SEF), Patrick Harms (UGOE), Steffen Herbold (UGOE), Andreas Hoffmann (FF), Christian John (FF), Libero Maesano (SEF), Martin Schneider (FF), Marc-Florian Wendland (FF)

**Reviewers**

Sergio Di Bona (DEDA), Claudia Di Napoli (CNR), Maurizio Giordano (CNR), Davide Guerri (DEDA), Raffaele Perego (CNR), Nicola Tonellotto (CNR)
# TABLE OF CONTENTS

**EXECUTIVE SUMMARY** ........................................................................................................... 3
**TABLE OF CONTENTS** ............................................................................................................... 15
**TERMS AND DEFINITIONS** ...................................................................................................... 17
**SYMBOLS AND ACRONYMS** .................................................................................................... 21

## 1 INTRODUCTION

1.1 PROJECT DESCRIPTION ........................................................................................................... 23
1.2 PURPOSE OF THE DOCUMENT ......................................................................................... 23
1.3 DOCUMENT STRUCTURE ....................................................................................................... 26

## 2 BACKGROUND

2.1 SOA AND SOA TESTING ....................................................................................................... 30
   2.1.1 Interface-based SOA ..................................................................................................... 30
   2.1.2 Contract-based SOA .................................................................................................... 31
   2.1.3 SOA model driven engineering, validation and verification ....................................... 34
2.2 MIDAS OBJECTIVES ............................................................................................................. 36
   2.2.1 Model-based test automation ..................................................................................... 37
   2.2.2 On demand test automation ..................................................................................... 42

## 3 GENERAL COMPATIBILITY REQUIREMENTS AND RECOMMENDATIONS

3.1 GENERAL REQUIREMENTS ON MODELS .......................................................................... 45
   3.1.1 Essential service platform independent model ............................................................. 45
   3.1.2 Service platform specific model .................................................................................. 45
   3.1.3 System under test model ............................................................................................. 46
   3.1.4 Test configuration model ............................................................................................. 47
   3.1.5 Test Context ................................................................................................................ 47
3.2 GENERAL REQUIREMENTS ON THE SYSTEM UNDER TEST ........................................... 48
   3.2.1 System under test technical interoperability platform ............................................... 48
   3.2.2 System under test initialization and recovery procedure ............................................. 48
3.3 GENERAL RECOMMENDATIONS ON THE SYSTEM UNDER TEST .................................. 49
   3.3.1 WS-Discovery implementation .................................................................................... 49
   3.3.2 WS-MetadataExchange and WS-Transfer implementation ....................................... 49
   3.3.3 Service Component Architecture (SCA) deployment ................................................ 49

## 4 COMPARABILITY RECOMMENDATIONS FOR FUNCTIONAL TESTING

4.1 FUNCTIONAL TESTING ........................................................................................................... 51
   4.1.1 Service unit functional testing .................................................................................... 51
   4.1.2 Service integration functional testing ........................................................................ 54
4.2 RECOMMENDATIONS ON MODELS FOR FUNCTIONAL TESTING .................................... 56
   4.2.1 Service interaction protocol model ............................................................................ 56
   4.2.2 Service function model ............................................................................................... 57
   4.2.3 Service choreography model ....................................................................................... 57
   4.2.4 Service composition model ......................................................................................... 58
   4.2.5 Service orchestration executable model (script) ......................................................... 58
4.3 RECOMMENDATIONS ON THE SYSTEM UNDER TEST FOR FUNCTIONAL TESTING .......... 59
   4.3.1 Resource state reinitialization service implementation .............................................. 59
   4.3.2 Transparency service implementation ....................................................................... 60

## 5 COMPARABILITY RECOMMENDATIONS FOR SECURITY TESTING

5.1 SECURITY TESTING ............................................................................................................... 61
5.1.1 Vulnerability Testing ................................................................. 61
5.1.2 Model-based fuzzing ............................................................... 62
5.2 RECOMMENDATIONS ON MODELS FOR SECURITY TESTING .............................................................................. 64
  5.2.1 Service interaction protocol model ............................................. 64
  5.2.2 Security test model .................................................................. 65
  5.2.3 Security policy machine readable model ...................................... 65
  5.2.4 Encryption/decryption and signature/validation algorithms and keys ....................................................... 65
5.3 RECOMMENDATIONS ON THE SYSTEM UNDER TEST FOR SECURITY TESTING ................................................. 65
  5.3.1 Resource state re/initialization service implementation ............... 65
  5.3.2 Transparency service implementation ........................................ 66

6 COMPATIBILITY RECOMMENDATIONS FOR USAGE-BASED TESTING ................................................................. 67
  6.1 USAGE-BASED TESTING ............................................................... 67
  6.2 RECOMMENDATIONS ON MODELS FOR USAGE-BASED TESTING ........................................................................ 68
    6.2.1 Usage journal for usage profile inference ................................... 68
    6.2.2 Recommended models for functional testing ................................ 69
  6.3 RECOMMENDATIONS ON THE SYSTEM IN THE FIELD FOR USAGE BASED TESTING .............................................. 69
    6.3.1 System in the field model compliance ........................................ 69
    6.3.2 Usage data observation .......................................................... 69
    6.3.3 Usage data accuracy ............................................................... 70
  6.4 RECOMMENDATION ON THE SYSTEM UNDER TEST FOR USAGE-BASED TESTING .............................................. 70
    6.4.1 Recommended enactments for functional testing ...................... 70

7 ANNEXES ......................................................................................... 71
  7.1 SERVICE ARCHITECTURE FOR HEALTHCARE – PILOT GENERAL DESCRIPTION ...................................................... 72
    7.1.1 The background (EHR, integration among care settings, continuity of care...) ............................................. 72
    7.1.2 The interoperability platform of Dedalus: X1.V1 ............................................................................... 75
    7.1.3 X1.V1 and the interoperability issues ............................................. 76
    7.1.4 Towards a SOA approach: the implementation of the HSSP services ....................................................... 76
  7.2 SERVICE ARCHITECTURE FOR SUPPLY CHAIN MANAGEMENT – PILOT GENERAL DESCRIPTION .............................. 81
    7.2.1 The background: Supply Chain Management in Context ................ 84
    7.2.2 The interoperability platform of Aragon Institute of Technology: ITChain (Information Technologies for Collaborative Supply Chain) ........................................... 86
    7.2.3 ITChain and the SOA & interoperability issues ............................ 88
    7.2.4 Towards a SOA approach: the implementation of Supply Chain Management Services ................................ 91
  7.3 EXAMPLE: THE INTERBANK EXCHANGE NETWORK ................................................................................................. 94
    7.3.1 Introduction .............................................................................. 94
    7.3.2 Essential service platform independent model ............................. 97
    7.3.3 System under test model ........................................................... 109
    7.3.4 Test configuration model .......................................................... 112
    7.3.5 Conclusion ............................................................................... 120

REFERENCES ....................................................................................... 123
### TERMS AND DEFINITIONS

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Application Programming Interface</strong></td>
<td>An application programming interface (API) is a protocol intended to be used as an interface by software components to communicate with each other. (<a href="http://en.wikipedia.org/wiki/Application_programming_interface">http://en.wikipedia.org/wiki/Application_programming_interface</a>)</td>
</tr>
<tr>
<td><strong>Black box testing</strong></td>
<td>A testing activity conducted without knowledge of the internal structure of the system under test [78].</td>
</tr>
<tr>
<td><strong>False negative</strong></td>
<td>Pass (test) verdict in presence of SUT failures.</td>
</tr>
<tr>
<td><strong>False positive</strong></td>
<td>Fail (test) verdict in absence of SUT failures.</td>
</tr>
<tr>
<td><strong>Generic service</strong></td>
<td>Service whose operation are generic, i.e. designated without specification of input and output types and that can be instatiated with user-supplied types</td>
</tr>
<tr>
<td><strong>Graphical User Interface</strong></td>
<td>A graphical user interface is a type of user interface that allows users to interact with electronic devices using images rather than text commands. (<a href="http://en.wikipedia.org/wiki/Graphical_user_interface">http://en.wikipedia.org/wiki/Graphical_user_interface</a>)</td>
</tr>
<tr>
<td><strong>Grey box testing</strong></td>
<td>Testing activity conducted with a partial knowledge of the internal structure of a system.</td>
</tr>
<tr>
<td><strong>Interceptor</strong></td>
<td>Test component that is virtually placed in a connection between two nodes of a SUT and is able to observe and analyze interactions and perform actions.</td>
</tr>
<tr>
<td><strong>Interface</strong></td>
<td>External structure of a system that allows the interaction with the system.</td>
</tr>
<tr>
<td><strong>Interoperability platform</strong></td>
<td>Technological platform allowing interoperability of systems whose implementations are a priori non-interoperable.</td>
</tr>
<tr>
<td><strong>Model Driven Architecture</strong></td>
<td>An Object Management Group initiative aiming to represent requirements and specifications as formal models at different levels of abstraction and to map these models to system implementations (<a href="http://www.omg.org/mda/">http://www.omg.org/mda/</a>).</td>
</tr>
<tr>
<td><strong>Model-driven engineering</strong></td>
<td>Software development methodology which focuses on creating and exploiting models and mapping and transforming these models to system implementations.</td>
</tr>
<tr>
<td><strong>Observation</strong></td>
<td>Test data reflecting the reactions from the SUT [78].</td>
</tr>
<tr>
<td><strong>Service</strong></td>
<td>A service is an activity that has an effect in the real/digital world, carried out by a system acting as a service provider for or in behalf of another system acting as a service consumer [69].</td>
</tr>
<tr>
<td>Term</td>
<td>Definition</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Service application programming interface</td>
<td>An application programming interface implemented on an interoperability platform.</td>
</tr>
<tr>
<td>Service choreography</td>
<td>Exchange of services in a services architecture for which an interaction protocol among participants is defined from a global perspective, each participant implementing its role without central control</td>
</tr>
<tr>
<td>Service choreography model</td>
<td>Model of the exchange of services in a services architecture for which an interaction protocol among participants is defined from a global perspective.</td>
</tr>
<tr>
<td>Service choreography testing</td>
<td>Testing the control flow of a service component architecture against a service choreography model.</td>
</tr>
<tr>
<td>Service composition</td>
<td>Relationship among service components of a service component architecture in which, in order to provide services, service components consume services provided by other service components.</td>
</tr>
<tr>
<td>Service composition model</td>
<td>Control and data flow model of a service composition that conforms to a formal meta-model.</td>
</tr>
<tr>
<td>Service composition testing</td>
<td>Testing a service composition against a service composition model.</td>
</tr>
<tr>
<td>Service component</td>
<td>A deployed software component implementing one or more service provider and consumer roles.</td>
</tr>
<tr>
<td>Service component architecture</td>
<td>Software technology that provides a implementable model for composing applications that follow Service-Oriented Architecture principles (<a href="http://oasis-opencsa.org/sca">http://oasis-opencsa.org/sca</a>).</td>
</tr>
<tr>
<td>Service contract</td>
<td>A specification of a service including: (i) the service operations; (ii) the service interfaces of the parties; (iii) the service parties external behaviors, including security and performance aspects.</td>
</tr>
<tr>
<td>Service contract model</td>
<td>A model of a service contract conforming to a formal meta-model</td>
</tr>
<tr>
<td>Service model</td>
<td>A service contract model or a services architecture model</td>
</tr>
<tr>
<td>Service orchestration</td>
<td>Exchange of services among service components in a service component architecture that is conducted (orchestrated) through the execution of an (orchestration) script by a service component playing the orchestrator role.</td>
</tr>
<tr>
<td><strong>Service orchestration script</strong></td>
<td>Script expressed in an executable script language that conducts a service orchestration.</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>--------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Service oriented architecture</strong></td>
<td>Service Oriented Architecture (SOA) is a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains [69]. SOA is an architectural paradigm for defining how people, organizations, and systems provide and use services to achieve results [63].</td>
</tr>
<tr>
<td><strong>Service Platform Independent Model</strong></td>
<td>Logical model, i.e. abstract computational model, of a service that is independent from a specific interoperability platform.</td>
</tr>
<tr>
<td><strong>Service Platform Specific Model</strong></td>
<td>Implementable model of a service on a specific interoperability platform.</td>
</tr>
<tr>
<td><strong>Service unit testing</strong></td>
<td>The testing of a service node of a deployed services architecture. If the service component is a composite one, it is tested as a whole.</td>
</tr>
<tr>
<td><strong>Services Architecture</strong></td>
<td>Network whose nodes are abstract system classes (participants) and whose links are services.</td>
</tr>
<tr>
<td><strong>Services architecture model</strong></td>
<td>Model of a services architecture including (i) the collection of Participant models, (ii) the collection of Service Contract models, and eventually the collection of control and data flow models.</td>
</tr>
<tr>
<td><strong>Emulator</strong></td>
<td>Test component that replaces virtually SUT nodes in a test configuration.</td>
</tr>
<tr>
<td><strong>Stimulus</strong></td>
<td>Test data sent to the SUT in order to control it and to make assessments about the SUT when receiving the SUT reactions to these stimuli.</td>
</tr>
<tr>
<td><strong>System In the Field</strong></td>
<td>The system in the field is a single-node or multi-node instantiated services architecture that is deployed and run in an operational environment</td>
</tr>
<tr>
<td><strong>System Under Test</strong></td>
<td>The system under test is a single-node or multi-node instantiated services architecture that is deployed and run in a testing environment</td>
</tr>
<tr>
<td><strong>Test arbitration</strong></td>
<td>Testing activity that assigns a test verdict to a test execution outcome.</td>
</tr>
<tr>
<td><strong>Test case</strong></td>
<td>Specification of how a set of test components interact with an SUT to realize a test objective. Test cases are owned by test contexts [78].</td>
</tr>
<tr>
<td><strong>Test component</strong></td>
<td>Test components are part of a test configuration and are used to communicate with the system under test (SUT) and other test components [78]. In MIDAS, test components are classified in three categories: proxy, emulator and interceptor.</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Test configuration</strong></td>
<td>Collection of test components and of connections between the test components and to the SUT. A test configuration is part of a test environment [78].</td>
</tr>
<tr>
<td><strong>Test configuration model</strong></td>
<td>Model of a test configuration conforming to a formal meta-model.</td>
</tr>
<tr>
<td><strong>Test context</strong></td>
<td>A test context groups a set of test cases and a test environment [78].</td>
</tr>
<tr>
<td><strong>Test environment</strong></td>
<td>A test environment is the composition of a SUT and an associated Test configuration [78].</td>
</tr>
<tr>
<td><strong>Test execution</strong></td>
<td>Testing activity that executes one or more tests and collects theirs outcomes for test arbitration.</td>
</tr>
<tr>
<td><strong>Test generation</strong></td>
<td>Testing activity that generates tests ready to be scheduled, executed and arbitrated.</td>
</tr>
<tr>
<td><strong>Test model</strong></td>
<td>Model of whose purpose is to drive and inform test generation, scheduling, execution and arbitration.</td>
</tr>
<tr>
<td><strong>Test scheme</strong></td>
<td>Test automation package that puts into operation test generation, scheduling, execution, arbitration following one or more specific testing methods</td>
</tr>
<tr>
<td><strong>Test scheduling</strong></td>
<td>Testing activity that selects one or more tests in a test collection for execution.</td>
</tr>
<tr>
<td><strong>White box testing</strong></td>
<td>Testing based on an analysis of the internal structure of the component or system [78].</td>
</tr>
<tr>
<td>SYMBOLS AND ACRONYMS</td>
<td></td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-----------------------------------------------------------------</td>
</tr>
<tr>
<td>ALM</td>
<td>Application Lifecycle Management</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>BPEL</td>
<td>Business Process Execution Language</td>
</tr>
<tr>
<td>BPMN</td>
<td>Business Process Model and Notation</td>
</tr>
<tr>
<td>CC</td>
<td>Common Criteria</td>
</tr>
<tr>
<td>CDA</td>
<td>Clinical Document Architecture</td>
</tr>
<tr>
<td>DICOM</td>
<td>Digital Imaging and COmmunications in Medicine</td>
</tr>
<tr>
<td>ETSI</td>
<td>European Telecommunications Standards Institute</td>
</tr>
<tr>
<td>EHR</td>
<td>Electronic Health Record</td>
</tr>
<tr>
<td>EPR</td>
<td>Electronic Patient Record</td>
</tr>
<tr>
<td>GP</td>
<td>General Practitioner</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>HL7</td>
<td>Health Level Seven</td>
</tr>
<tr>
<td>HSSP</td>
<td>Healthcare Services Specification Program</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated Development Environment</td>
</tr>
<tr>
<td>IG</td>
<td>Implementation Guide</td>
</tr>
<tr>
<td>IHE</td>
<td>Integrating the Healthcare Enterprise</td>
</tr>
<tr>
<td>JSON</td>
<td>JavaScript Object Notation</td>
</tr>
<tr>
<td>LOINC</td>
<td>Logical Observation Identifiers Names and Codes</td>
</tr>
<tr>
<td>MDA</td>
<td>Model-driven Architecture</td>
</tr>
<tr>
<td>MDE</td>
<td>Model-driven Engineering</td>
</tr>
<tr>
<td>MIDAS</td>
<td>Model and Inference Driven Automated testing of Services</td>
</tr>
<tr>
<td>NAV</td>
<td>Notification of Document Availability</td>
</tr>
<tr>
<td>OASIS</td>
<td>Organization for the Advancement of Structured Information</td>
</tr>
<tr>
<td>Standards</td>
<td></td>
</tr>
<tr>
<td>OCL</td>
<td>Object Constraints Language</td>
</tr>
<tr>
<td>OMG</td>
<td>Object Management Group</td>
</tr>
<tr>
<td>PCCR</td>
<td>Patient Care Coordination Report</td>
</tr>
<tr>
<td>PDQ</td>
<td>Patient Demographics Query</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>PHR</td>
<td>Personal Health Record</td>
</tr>
<tr>
<td>PIX</td>
<td>Patient Identifier Cross-Referencing</td>
</tr>
<tr>
<td>PIM</td>
<td>Platform Independent Model</td>
</tr>
<tr>
<td>PS</td>
<td>Patient Summary</td>
</tr>
<tr>
<td>PSM</td>
<td>Platform Specific Model</td>
</tr>
<tr>
<td>REST</td>
<td>Representational state transfer</td>
</tr>
<tr>
<td>SaaS</td>
<td>Software as a Service</td>
</tr>
<tr>
<td>SAML</td>
<td>Security Assertion Markup Language</td>
</tr>
<tr>
<td>SCA</td>
<td>Service Component Architecture</td>
</tr>
<tr>
<td>SFR</td>
<td>Security Functional Requirements</td>
</tr>
<tr>
<td>SIF</td>
<td>System In the Field</td>
</tr>
<tr>
<td>SME</td>
<td>Micro, Small and Medium-sized Enterprise</td>
</tr>
<tr>
<td>SOA</td>
<td>Service Oriented Architecture</td>
</tr>
<tr>
<td>SoaML</td>
<td>Service oriented architecture Modeling Language</td>
</tr>
<tr>
<td>SOAP</td>
<td>W3C messaging standard [64][65][66][67]</td>
</tr>
<tr>
<td>SUT</td>
<td>System Under Test</td>
</tr>
<tr>
<td>TEaaS</td>
<td>Test Environment as a Service</td>
</tr>
<tr>
<td>TSM</td>
<td>Test Scheme Management</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modeling Language</td>
</tr>
<tr>
<td>UTP</td>
<td>UML Testing Profile</td>
</tr>
<tr>
<td>WS</td>
<td>Web service</td>
</tr>
<tr>
<td>WSDL</td>
<td>Web Service Definition Language</td>
</tr>
<tr>
<td>WS-I</td>
<td>Web Services Interoperability Organization</td>
</tr>
<tr>
<td>XML</td>
<td>eXtensible Markup Language</td>
</tr>
<tr>
<td>XDS</td>
<td>Cross-Enterprise Document Sharing</td>
</tr>
<tr>
<td>XSD</td>
<td>XML Schema</td>
</tr>
</tbody>
</table>
1 INTRODUCTION

1.1 Project description

The objective of the MIDAS project is to carry out a comprehensive framework able to support the automation and intelligent management of SOA core testing tasks, such as test generation, scheduling, execution and arbitration. Furthermore, the framework supports all the testing cycle activities: test case planning, development and execution, reporting and result analysis, test campaign management and scheduling. Moreover, the framework enables advanced functional, security and usage-based testing methods.

In order to provide these features, the architecture of the MIDAS framework includes:

- an environment for design time and run time (on the fly) model-based generation of test cases and oracles;
- an environment for SOA automatic testing configuration, initialization and execution, based upon a Test and Test Control Notation (TTCN-3) engine;
- probabilistic and symbolic inference based methods and tools for test scheduling;
- usage profile inference based on usage observation journals collected on the system in the field.

The end user service model of the MIDAS framework is Software as a Service (SaaS). In order to support the elasticity of the testing environment – allocation of substantial amounts of computation, memory and bandwidth resources for relatively short test campaigns on large services architectures – the MIDAS framework is deployed on a Cloud infrastructure.

In order to evaluate the effectiveness and the usability of the MIDAS framework facilities, two pilot SOA testing experiences are carried out in different business domains: Healthcare (HC) and Supply Chain Management (SCM).

1.2 Purpose of the document

The purpose of this document is to state the requirements and the recommendations for SOA testing and usage observation with the MIDAS on demand test automation facility. They concern specifically the employ
of: (i) the MIDAS core testing functionalities that are the automated test generation, scheduling, execution and arbitration for functional, security and usage-based testing, and (ii) the MIDAS core usage observation functionalities for usage data collection and usage profile modeling.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [54].

We will employ the generic term ‘System Under Test’ (SUT) to refer both to single-node service providers and multi-node services architecture under test, where the distinction is not relevant. Similarly, we will employ the generic term ‘System In the Field’ (SIF) to refer both to single-node service providers and multi-node services architecture that are deployed and run in the operational context, and are the targets of the MIDAS usage observation facilities.

The MIDAS testing approach is non-intrusive on the SUT. Only the connections between the SUT stimuli, response, observation and state initialization points and the MIDAS facility must be established in order to run tests.

MIDAS test compatibility requirements must be fulfilled by any SUT that is sought to be a MIDAS testing target. With respect to MIDAS compatibility requirements, our work has been driven by two methodological principles: (i) minimize their number and their enactment burden for the user; (ii) formulate only requirements whose fulfillment improves the dependability, security, interoperability, conformance with standards and, last but not least, auditability and testability as a generally accepted criteria - non-specific to MIDAS - of single-node service providers and multi-node services architectures.

The MIDAS test facility proposes its functionalities as a portfolio of test schemes. Each test scheme is a collection of implemented behaviors and data structures that automates test generation, scheduling, execution and arbitration functionalities related to a specific testing approach, method and practice (e.g.: systematic functional testing, fuzz testing, etc.). A test scheme is developed and installed on the MIDAS facility, following a standard procedure and format, by a test scheme developer, and is invoked by the end user. The general idea is that the end user invokes several test schemes on a SUT to perform multi-approach automated test generation and running.
MIDAS compatibility recommendations are classified as general vs. test scheme specific.

General recommendations are related to the use of features of the MIDAS facility that are independent from specific test schemes. The use of these features is optional, but the fulfillment of the related recommendation is a prerequisite for the use of these features.

Test scheme specific recommendations have to be adopted only if the user wants to invoke the specific test scheme on the SUT. Obviously, the invocation of a specific test scheme is optional for the user, but if the user wants to use it the fulfillment of the related recommendations will be a prerequisite. Moreover, MIDAS usage observation and profiling are optional for the end user, but the related compatibility recommendations are prerequisites for usage-based testing with MIDAS. Anyway, the MIDAS facility proposes some basic test schemes that will be operational without requesting the satisfaction of any specific recommendation - only the general compatibility requirements must be fulfilled.

The MIDAS testing approach is model-based. In order to perform automated testing on the SUT and to support usage observation on the SIF, the MIDAS facility needs the upload of the appropriate model artifacts.

This document uncovers the MIDAS compatibility requirements and recommendations that concern: (i) the upload of model artifacts - including the specification of their meta-models - that are needed to drive the automated test generation, scheduling, execution and arbitration, in general and within specific test schemes, on the SUT, and to support the usage observation on the SIF; (ii) the SUT/SIF configuration and deployment, in order to consent respectively the binding, connection and interactions with the MIDAS facility through the MIDAS SUT interface and the installation of usage observation facilities and the collection of usage data on the SIF.

At the delivery date of this document, the requirements, specifications and architecture of the MIDAS facility are still work in progress. The outcomes of this work may change some of the compatibility requirements and recommendations expressed therein. In
any case, this document must not be considered a MIDAS user guide as such, but constitutes the basis on which the user guide will be built\(^{12}\).

The first recipients of this document are the MIDAS early adopters, such as the MIDAS project partners that are in charge of the pilots (Healthcare Generic Services pilot / Supply Chain Management pilot).

Early adopters will act as users of the MIDAS on demand test and usage observation automation facility, in their roles of testers, i.e. users of the MIDAS facility for SUT testing, testees, i.e. craftsmen of the SUT deployments for testing with the MIDAS facility, and supervisors, i.e. administrators of the SIF operation that seek usage observation. This document will constitute the support that allows them to start trying the fulfillment of the MIDAS compatibility requirements and recommendations on their SUTs and SIFs. Relatively to these points, the feedback from the MIDAS pilots’ experience will be precious all along the project. If some of the MIDAS compatibility requirements and recommendations will change as a consequence of design and implementation choices and of pilots’ feedbacks, updated versions of this document including these changes will be made available.

1.3 Document structure

The section 2 (Background), is split in two parts. In the first part (2.1 SOA and SOA testing) we introduces in a concise way the specific traits of the MIDAS facility target, i.e. service and service architecture implementations, and the challenges that this kind of target poses to model-driven engineering, validation, verification and testing.

The MIDAS target does not restrict the kind of service and service architecture implementations that can be tested, from mere interface-based to full contract-based services, that are sketched in the paragraphs 2.1.1 and 2.1.2, the only prerequisites being the fulfillment of the compatibility requirements expressed in this document. The non-obvious relationships between services and services architectures engineering (whether model-driven or not), validation and verification and the specific role of testing in the process are concisely analyzed in the paragraph 2.1.3.

\(^{12}\) Apart from the general compatibility requirements, a user guide should expose the matter starting from a classification of user testing and usage observation needs and problems, then suggesting the appropriate test schemes and enunciating the related recommendations.
The second part of the section 2 (2.2 MIDAS objectives) gives an overall presentation of the MIDAS on demand test automation facility features.

The compatibility requirements - that must be fulfilled in order to test a services architecture with MIDAS - and recommendations – that should be satisfied to submit the testing target to specific checks and testing methods – are documented in the sections 3, 4, 5, 6.

Section 3 (General compatibility requirements and recommendations) presents the compatibility requirements that each MIDAS potential target must satisfy and some recommendations whose fulfillment improves the general testing process - and also the reliability and maintainability of the target services architecture. MIDAS approach is strongly model-based, hence requirements and recommendations are arranged in two groups: (i) those that bear on models that must (or should) be supplied and (ii) those concerning the configuration and the deployment of the system under test. These two groups are documented in two separate subsections. Furthermore, this arrangement is maintained in all the other sections.

The mere fulfillment of compatibility requirements allow basic testing methods (generally speaking, basic robustness testing methods) to be applied to the system under test. To be able to employ more enhanced testing methods and approaches, MIDAS requests as a prerequisite the fulfillment of specific recommendations. Sections 4, 5, 6 exhibits the same structure: the recommendations, categorized as bearing either on models or on the system configuration and deployment, are preceded by paragraphs on the principles of, respectively, functional, security and usage-based testing that try to explain and justify the following recommendations.

Section 4 (Compatibility recommendations for functional testing) concerns functional testing, specifically functional unit testing (testing one-node’s functions) and functional integration testing (testing the correct functional cooperation in a multi-node architecture). Recommendations are arranged following the crescendo of testing power that is allowed by their fulfillment, which coincides with the increase of the burden for the user, in terms of additional more sophisticated models to be supplied and of ancillary operations to be implemented on the system under test.
Section 5 (Compatibility recommendations for security testing) concerns security testing, more specifically vulnerability testing, i.e. challenging the presence of flaws that, even in the absence of compliance failures, allow malicious attacks. The methods chosen for that purpose is model-based fuzzing. Basic data fuzzing does not require the fulfillment of any specific recommendation, while behavioral fuzz testing requests the availability of some models, such as protocol models.

Section 6 (Compatibility recommendations for usage-based testing) presents the recommendations that should be satisfied in order to employ the usage-based testing approach. The general idea is that the usage of a services architecture *in the field* is a source of data, information and knowledge that can be used to influence the testing strategy and activity. The recommendations concern essentially the usage observation activity, i.e. the information from the system in the field that MIDAS needs to support the user with generation of lean usage observation software components (that should be deployed on the system in the field) and usage data organization.

In section 7 (Annexes) we give two overviews of the business domains and technical environments in which the MIDAS Pilot services architectures shall be put in place, and we develop an example that illustrates some core concepts presented in this document.

Subsection 7.1 (Service architecture for Healthcare – pilot general description) gives an overview of the spread of the SOA approach in healthcare and of the products and applications - developed and marketed by Dedalus S.p.A. and its partners - that will ground the Pilot services architecture for healthcare as an early target of the MIDAS on demand automated test facility.

Subsection 7.2 (Service architecture for Supply Chain Management – pilot general description) gives an overview of the Supply Chain Management domain and of the services architecture promoted by the Instituto Tecnologico de Aragon and that will be the basis of the Pilot services architecture for supply chain management as an early target of the MIDAS on demand automated test facility.

Subsection 7.3 (Example: The Interbank Exchange Network) illustrates some of the core concepts that are drawn in this document, in particular relatively to the “architectural” models whose availability is a general requirement for testability (3.1 General requirements on models), through an example. The example is a service oriented perspective of that
presented in the Annex D of the OMG UML Testing Profile V.1.2 specification [78].
2 BACKGROUND

2.1 SOA and SOA testing

SOA design and implementation style is currently practiced following approaches that are situated in a range between two extremities: (i) “weak” interface-based SOA and (ii) “strong” contract-based SOA [63].

The “weak” interface-based SOA style is based merely on the design of an Application Programming Interface (API) as a primary access device to some system functionalities and on the management of this interface as a “first-class” artifact that is separate from the system internal implementation.

In the contract-based “strong” SOA style, beyond separation between interface and implementation and hiding of systems’ internals that are a must in any case, a service is represented by an artifact referred to as a service contract that incorporates the formal definitions not only of the service interface, but also of the service function and the external behaviors of the parties, including security and performance aspects. The service contract acts as an agreement between the parties and as a collection of requirements for the implementers.

2.1.1 Interface-based SOA

An Application Programming Interface (API) is an artifact that is exposed by a system acting as a service provider and used by one or more systems acting as service consumers. The use of the API is the only means that allows the consumers to interact with the provider to coordinate the service provision/consumption.

In most cases, APIs are designed inside-out, as the result of exposing some system’s internal operations as service provisions. Frequently, the API does not represent a disembodied service – that can be provided by several different organizations and systems – and, even if managed as a separate artifact, it is bound to the unique concrete implementation that exposes it.

The spread of interface-based SOA is revealed by the impressive growth of the “API economy”¹³, pioneered by the Amazon Web Services¹⁴.

---

¹⁴ http://aws.amazon.com/ - the API approach has been “introduced” some years ago by Jeff Bezos (CEO of Amazon) when he said that “all IT assets are to be exposed as APIs”.

MIDAS • Project Number 318786 • D2.1 • Page 30 of 129
initiative. APIs are a key growth driver for hundreds of companies across a wide range of industry sectors and are going to be a primary access channel to technology-driven products and services.\footnote{The emblematic manifestation of this trend is the fact that, in 2012, Salesforce.com (http://www.salesforce.com/) has generated more than half of its $2.3 billion in revenue through its APIs, not through its browser interfaces. Twitter is said to process 13 milliard transactions per day through its APIs, and Amazon is rapidly closing in on a one thousand milliard transactions per day.}

Moreover, APIs, as separate “first-class” artifacts, push the growth of a new business sector: the API management services market. Start-ups and established companies offer to API players new services such as: (i) automation and control of the connections between the APIs and the applications that use them; (ii) API multi-version and multi-implementation management; (iii) monitoring of the traffic from consumer applications; (iv) support for memory management and caching mechanisms; (v) API access control and security management.

We consider that accessible, affordable and seamless black-box testing and usage observation of API service providers will be essential constituents of the next generation of API management services.

\subsection*{2.1.2 Contract-based SOA}

A service contract is an artifact that is intended to supply a formal description (more or less complete) of the service interfaces – the external structures that the parties expose each other to interact - but also of the service functions - what the service provider is intended to carry out for or in behalf of the service consumer - and of the external behaviors of the parties, visible at the interfaces, related to the coordination of the service provision/consumption (or its denial), including security and performance aspects.

The service contract formulates a black-box model of the system that intends to act as a service provider. As such, it conveys a functional - in the broad sense - perspective of the system and says nothing about its internal architecture and implementation.

From its inception, the contract-based SOA style is entrenched in the OMG Model Driven Architecture (MDA)\footnote{The main goals of the MDA initiative are: (i) to represent requirements and specifications as formal models at different levels of abstraction and (ii) to map these models to system implementations allowing automated transformation (http://www.omg.org/mda/index.htm).} initiative. As a result of the
application of the MDA approach, the OMG SoaML specifications [63] supplies model elements to represent: (i) the service contract (Service Contract), and (ii) the services architecture (Services Architecture), as an architecture of systems (Participant) bound by service contracts.

A Services Architecture formulates a white-box model that conveys a high level ontological perspective of a distributed system. As such, it captures not only the construction (i.e. the component sub-systems and their service relationships) but also, through additional control and data flow specifications (choreographies), the operation (i.e. the manifestation of the construction in the course of time) of a distributed system, while abstracting from implementation details.

The model-driven service orientation applied to system engineering allows the recursive and alternate application of the functional and ontological perspectives, formulated respectively as black-box and white-box models.

Starting from the top-level, the system functional perspective is modeled with the collection of Service Contracts that it fulfills as a provider and as a consumer. Its ontological perspective is modeled as a Services Architecture that allows specifying easily: (i) the system composition, i.e. the set of Participants that are inside its boundary; (ii) the system environment, a disjoint set of Participants that have direct service relationships with the composition Participants - the composition and the environment are separated by the system boundary; (iii) the system structure, i.e. the Service Contract relationships among the composition and environment Participants.

For each Participant, the recursive and alternate application of the functional and ontological perspectives described above can be repeated until any further decomposition is non-interesting or impractical [18].

An example of a service model, with samples of Participants, Service Contracts, Services Architectures and functional/constructional decompositions is presented in the section 7.3 (Example: The Interbank Exchange Network).

In conformance to MDA, the service model has a layered structure split in: (i) the conceptual layer, (ii) the logical layer and (iii) the implementable layer.

The conceptual layer includes formulations for service contracts and services architectures at the business level that are understandable by
non-technical people, such as the business stakeholders. As we will see below, the conceptual layer is relevant for service model validation, but not for model-based SOA testing as a verification method.

The logical layer accommodates the service platform independent model (service PIM), an abstract computational model [77][51][63][53] of service contracts and services architectures that is independent from specific technical platforms.

The implementable layer includes one or more service platform specific models (service PSMs) that are models of the services on specific technical platforms, such as SOAP [64][65] and REST [25], expressed by machine readable artifacts (WSDL; XSD; BPEL; WS-Policy; WADL; JSON)[83][85][104][105][106][107][7][8][95][96][97][80][36][37]. The task of these technical platforms is to ensure the technical interoperability of systems that are built on a priori non-interoperable implementation platforms. The elements of the service PSM are machine-readable artifacts (e.g. WSDL/XSD) that are used by the implementation platforms (JEE, .NET, C++, Python, etc.) to put in place the interoperable infrastructure of the service interaction.

Generally speaking, the service model includes the service PIM and the service PSM. The APIs produced through the “weak” interface-based SOA style are indeed service PSMs, i.e. machine-readable models of the service interfaces on specific interoperability platforms. We can say that the interface-based SOA design style produces only service PSMs.

The spread of contract-based SOA is probably two orders of magnitude less than that of interface-based SOA. Contract-based SOA style follows the diffusion of the model-driven engineering approach [4][38][59] and concerns today a growing number of distributed system architectures in critical business sectors. The contract-based SOA style starts to embrace “new” sectors (such as healthcare) where it is often linked to standardization initiatives\(^{18}\). Moreover, in several services

\(^{18}\) The Healthcare Services Specification Program (HSSP) - http://hssp.wikispaces.com/ - is a joint HL7/OMG that aims to standarize generic service contracts that are intended to regulate the service relationships on basic healthcare functionalities (such as the patient record management, the patient identity management, the terminology management, the healthcare setting directory management, the decision aid service management, and others). For each service, the HSSP standardization cycle distinguishes between the Service Functional Model (SFM) – a platform independent model, at the logical level – and the Service Technical Model (STM) – a platform specific model at the implementable level. An objective of the MIDAS project is to build through the MIDAS facility a testing framework for HSSP services.
architecture, the contract-based style gets along with the interface-based one: the former is generally reserved to the “core” service exchange whereas the latter concerns rather the “last mile” service access\textsuperscript{19}.

2.1.3 SOA model driven engineering, validation and verification

Model-driven engineering of services and services architectures changes fundamentally not only the design and implementation cycle – with the so-called contract-first approach\textsuperscript{[41][111]} - but also the validation and verification tasks.

Thanks to the model-driven approach, the canonical validation question that is expressed as: “Are you building the right services and services architecture?”, can split into two sub-questions that are related but consent separate answers: (i) a more specific validation question: “Are you establishing the right service model?” and (ii) a verification question: “Are you building the services architecture and the participant systems’ implementations right?”.

The service model acts as an agreement between the parties and a collection of requirements for the implementers. As such, it is the pivot of the model-driven validation and verification process. The former question is about the compliance of the service model with the stakeholders’ exigencies and constraints. The latter one is about the compliance of the implementations with the validated service model and the fulfillment of security and dependability criteria\textsuperscript{20}.

The validation the service model against the stakeholders’ exigencies and constraints is a critical and hard challenge. It is outside the scope of model-based automated testing and of the MIDAS project in particular.

According to the SOA approach, we assume that the white-box perspective of a system is the services architecture involving its sub-systems as participants. The only applicable verification methods are

\textsuperscript{19} For instance, the access to services from smart phones and other mobile devices.

\textsuperscript{20} Some dependability and security exigencies and constraints can be expressed as explicit contract clauses (for instance, the confidentiality of the service interactions contents). The verification question challenges the compliance of the implementation with the contract. Other very important exigencies are implicit “negative” clauses, for instance the absence of flaws that expose the system to malicious attacks. The verification question is about the implementation quality (robustness). A service provider implementation can be at the same time compliant with the functional and even security contract clauses, and vulnerable to malicious attacks.
model checking of specified control and data flows (choreographies) and grey box testing of these flows, i.e. of visible interactions between sub-systems seen as black-boxes.

When the white-box perspective is no more practicable with a sub-system, it appears as a black-box, and only the functional perspective - the services that it provides and consumes - is available. Classical white-box verification approaches, such as program formal analysis, code peer review and structural testing are no more workable (except by system owners). The only applicable verification method is black-box testing.

In summary, the main goals of the SOA testing activity are: (i) compliance testing - challenging the compliance of SUT with the service model (service contracts, services architecture, choreographies), including functional, security and performance aspects; (ii) vulnerability testing – challenging the absence of weaknesses of the SUT architecture and implementation that expose it to malicious attacks.

Effective testing procedures and processes are indispensable and powerful means for strengthening the confidence of the stakeholders in the functional and non-functional dependability and security of service providers and services architectures.

But SOA testing has the paradoxical trait that the same peculiarities that make it necessary make also it hard [1][3][6][12]: lack of observability of the involved systems, lack of trust in the employed engineering methods, lack of direct control of the lifecycles, late binding of systems, fundamental uncertainty of the test verdicts, organizational complexity, elastic demand of computational resources, increasing scale factor of the services architectures, high costs and, last but not least, questionable efficacy of humans in performing manually a more and more hard and complex but boring and low rewarding activity such as testing.

The consequence is that only big organizations, in critical sectors such as nuclear plants, telecom, transportation … are able to mobilize substantial human, hardware and software resources and carry out systematic testing activities on vital systems, for which validation and verification are mandatory.

In effects, hand-writing of test cases, manual configuration of test environments, manual scheduling of test runs and eyeball assessment of large numbers of test outcomes are not only labor intensive and difficult to put in practice, but also, even if it were possible to put them in practice, the least effective solution of the SOA testing problem.
The lack of affordable and accessible SOA testing facilities excludes small actors such as SMEs and start-up service developers and providers from testing activities that are nevertheless necessary to engineer sustainable service deployments and provisions. Many of the new digital processes in sectors such as healthcare are just as vital as those involved in the aforementioned “traditional” critical sectors, and their functional and non-functional dependability and security demand at least equally rigorous and intensive testing practices.

It is evident that the SOA testing challenges cannot be overcome by mere methodological recommendations on engineering practices, because of the cost and complexity of their implementations. The solution of the SOA testing and usage observation problems can be brought only by a disruptive innovation [15] that drastically simplifies and routinizes the testing task by realizing and offering an automated, effective, accessible and affordable testing facility.

2.2 MIDAS objectives

The goal of the MIDAS project is to design and build a testing automation facility that targets SOA implementations. The MIDAS facility is available on demand, i.e. on self-provisioning, pay-per-use, elastic basis. The MIDAS facility will be a multi-tenancy SaaS (Software as a Service) deployed on a cloud infrastructure and accessible in the Internet.

The targets of the MIDAS facility are both “weak” interface-based and “strong” contract-based SOAs. The MIDAS facility is intended to provide: (i) automation of the “core” testing activities: test generation, scheduling, execution and arbitration, and (ii) support for the usage observation enactment and data collection.

The MIDAS test facility shall put into operation automated black-box testing of single-node SUTs (service unit test) and grey-box testing of multi-node SUTs, where the connections between the nodes are observable (service integration test). In any case, it shall not be intrusive on the implementation under test. The MIDAS usage observation facility shall support usage observation component generation and usage data collection - the direct installation of the observation components and collection of usage data on the SIF being outside the scope of the MIDAS project.

MIDAS test automation is based on models. The underlying idea is that the activity of the SOA tester and testee shall shift from test case
hand-writing, manual configuration of test environments, manual scheduling of test runs and eyeball assessment of test outcomes, to model authoring. The burden of test generation, scheduling, execution and arbitration, until the fabrication of the test report, is given up to the MIDAS facility. MIDAS usage-based testing is based on models too: models support the generation of the observation components, the arrangement of usage journals and the generation of usage profiles that are integrated in testing strategies.

The MIDAS test facility will support functional and security testing. Performance testing is out of the scope of the MIDAS project, but the MIDAS architecture will be able, through the mechanism of installation of new test schemes, to accommodate easily further performance testing approaches, practices and methods.

The availability on the MIDAS facility of model authoring tools is still an open project issue. In any case, it shall be easy to import in the MIDAS facility compatible models from external user-preferred model authoring tools.

### 2.2.1 Model-based test automation

The model-based testing and usage observation approach in MIDAS is based on two different kinds of models:

(i) **system models**,  
(ii) **test models**.

#### 2.2.1.1 System model

The system model is a pure model of the MIDAS target system, i.e. it represents an abstraction of the target system that is independent of any test scheme. It includes two distinct sub-models:

- **the service model** - the model of the services and of the services architecture that are put into operation within the SUT, at the **logical level** (service PIM) and at the **implementable level** (service PSM);  
- **the SUT model** - the description of the deployed and accessible SUT topology and locations.

The structure of the service model is presented in the preceding 2.1.2 section (Contract-based SOA). Generally speaking, service models are authored within the model-driven, contract-first service engineering
cycle, or by reverse modeling from the service implementations and their behaviors.\footnote{Notations for platform independent and platform specific reverse modelling are standardized by the OMG Architecture-Driven Modernization Task Force (http://adm.omg.org/).}

The MIDAS facility requires, for each SUT, an essential service PIM that includes the Services Architecture of the SUT and the involved Service Contracts and Participants skeletons. The Service Contract skeleton identifies the service interfaces related to the contract roles, and the Participant skeleton identifies the Service Contract roles that it fulfills. Obviously, richer service PIMs may include interaction protocols (UML Interaction), participant state-charts (UML Protocol State Machine), choreographies (UML Activity), operation semantic specifications (OCL pre/post-conditions), and other modeling elements, such as security specifications [53]. All these elements can be used by specific test schemes, but they are not required as elements of the essential service PIM.

The essential service PIM has pointers to the associated service PSM elements (the collection of machine readable artifacts such as WSDL, XSD ...). For instance, the Service Contract interfaces refer to WSDL artifacts as PSMs on the SOAP interoperability platform. Note that only service PSM artifacts (APIs) are available for interface-based SOA style services. In this case, and in any case in which a service model is not available, the MIDAS user must author a Service Contract skeleton for each deployed API and a Services Architecture that includes the Service Contract skeletons and the Participant skeletons that fulfill them. Hence, the MIDAS user can apply directly to the SUT all the test schemes that do not demand more elaborated service models or test-related models (for instance, there are basic test schemes that use only the service PSM, i.e. the API syntax, such as data fuzzing)\footnote{The MIDAS user must author also the SUT model and the Test Configuration model that shall be supplied to the test profile generation and running plug-ins.}.

The essential service PIM includes also the definition of auxiliary “relay” Participants. For each Service Contract is defined a “relay” Participant that realizes both (an only) the service interfaces of the contract parties. These auxiliary Participants model relays logically placed on the communication paths that convey the associated service interactions.
The essential service PIM is a class model, and its elements are used as *classifiers* by the other models such as the SUT model and the test configuration model. From the testing and usage observation points of view, it is a *descriptive* model of the deployed and accessible target system, and its accuracy is crucial, because it supplies elements (its classes) that are used to instantiate test model elements that drive the automatic generation of components of the test execution system.

Samples of the essential service PIM elements related to the Interbank Exchange Network example are presented in the section 7.3.2 (Essential service platform independent model).

The SUT *model* describes the *topology* of the deployed accessible SUT – the deployed *nodes* that are accessible, their accessible *ports* where are exposed the service *provided* and *required interfaces* and the declared *communication paths* between the ports. The SUT model also documents the *locations* of its accessible node/ports.

The formulation of the SUT model is based on UML *Deployments*. The SUT nodes are represented by UML *Node* instances that are classified by the service model *Participants* and their *ports* by the services interfaces (*Provider*, *Consumer*, *Service Interface* classifiers).

The SUT model is supplied by the MIDAS user to be uploaded onto the MIDAS facility. The deployed SUT nodes, ports and communication paths that are *identified*, *described* and *located* by an uploaded SUT model are made *accessible* by the MIDAS facility as such. Conversely, actually deployed SUT nodes and ports that are not described in the SUT model are not accessible by the MIDAS facility.

We present in the section 7.3.3 (System under test model) details and samples of the SUT model related to the Interbank Exchange Network example.

### 2.2.1.2 Test model

The *test model* is a model of a testing practice, method and scenario to be applied to the SUT. It is embodied in a *Test Context* that conforms to the UML Testing Profile [78]. In the MIDAS facility, a UTP *Test Context* is automatically created when the MIDAS user invokes a *test scheme* and is

---

[23] UML Testing Profile V. 1.2 is still in the beta 2 version when this document is delivered.
initialized by the test scheme implemented methods. The test context is the pivot of the test model and bears the specifications of a test scenario.

The *Test Context* aggregates the *test configuration model* (TC model), the *Test Cases* that are brought by the test scheme and, eventually, one or more *test-related models*. The TC model specifies what parts of the SUT are the targets of stimuli, responses and observations. Specific test schemes use, in addition to system models, *test-related models*. Generally speaking, test-related models aim to represent test ends, means, such as directives and course of actions, and are related to the test methods that use them.

On the basis of the SUT model, the test configuration model (TC model) *prescribes* the architectural configuration of a test scenario in a manner that is independent by the test scheme that utilizes it. It indicates: (i) the SUT accessible nodes that shall be the targets of stimuli, responses and observations in the course of the test run, (ii) the SUT accessible places - ports and communications paths - where the stimuli, responses and observations shall be acted by the MIDAS facility and (iii) the connections that shall be established between SUT and the MIDAS test execution system.

A TC platform independent model (PIM) is produced by a user-driven model-to-model (M2M) transformation of the SUT model. The formulation of the TC model is based on UML *Component* [77] and UTP [78] extended by the MIDAS UML Profile and Meta-model. Its elements are classified by the essential service PIM. The TC model is a *prescriptive* model: it drives the generation of the architectural components of the test execution system.

The SUT deployed and accessible nodes, that are modeled as UML *Nodes* instances (with their accessible ports) in the SUT model, whose behavior is the target of stimuli, responses and observations in the test scenario, are modeled in the TC model as *Proxy* instances. Each *Proxy* instance is classified by the *Participant* that classifies the corresponding SUT model *Node* instance. Each *Proxy* instance drives the automated generation of the test execution system component that is responsible for the binding, connection and interaction between the MIDAS test execution system and the corresponding node of the SUT.
An Emulator instance models a MIDAS test execution system component that is able to emulate either a SUT deployed node or an environment virtual node and its ports\(^{24}\) that interacts with its interlocutor SUT nodes through their representative Proxy instances. An Emulator instance is classified by the corresponding Participant and, as such, models only its service interaction capabilities that are instantiated from its Participant classifier.

An Interceptor instance models a MIDAS test execution system component that is able to behave as a logical relay that is placed on an accessible SUT communication path. The modeled test execution system component is able to capture, observe and retransmit the interactions between the nodes at the communication path endpoints. An Interceptor instance can be placed only on the communication path between two Proxy instances and implements only the service interaction capabilities that are instantiated from its relay Participant classifier.

The testing operational semantics of the Emulators and the Interceptors is unspecified by the TC model and is provided by the test scheme that is invoked by the user through its Test Cases. A test scheme that is compliant with the MDA approach performs a two-step generation of test execution system: (i) in the first step it produces a platform independent UTP Test Context by M2M transformation from its Test Cases and the TC PIM; (ii) in the second step it generates the code artifacts for a specific test execution platform (such as the TTCN-3 test execution environment) by M2T transformation of the Test Context.

A SUT model can be associated with several TC PIMs, each representing a different test configuration scenario on the SUT. Each TC PIM can be used by different test schemes. In the MIDAS project, the target specific test execution platform is the TTCN-3 execution testing environment, but the MIDAS model-driven approach enables automatic generation of test execution code artifact on other test execution platforms.

---

\(^{24}\) The ontological model of the system that is a target of the testing activities distinguishes between the composition nodes and the environment nodes. The former are part of the system while the latter are the nodes outside of the system boundary that interact directly with the former. Only the former are physically deployed in the SUT and the latter must be emulated by the test execution system in order to emulate the interaction of the SUT with its environment. An Emulator can also model a test execution system component that emulates a deployed SUT node, whose behaviour is not under consideration in the test scenario.
The TC model, the MIDAS Profile and Meta-model elements and some test configuration patterns are presented and illustrated through samples from the Interbank Exchange Network example in the section 7.3.4 (Test configuration model).

Test schemes may use, in addition to the TC model and their own Test Cases, a family of models generally referred to as test-related models that aim to represent test ends, means, directives and courses of action. They drive and inform the automated test generation, scheduling, execution and arbitration performed by the implemented behaviors of the test schemes. They are authored by the testers and other professionals involved in the verification tasks.

There are different types of test-related models such as fault models, vulnerability models and others. Elaborate testing methods such as advanced fault-based testing combine test goal models (for instance, covering the presence of a priori anticipated faults) with the service models, in order to filter and prioritize compliance test cases that can be systematically generated by the service models.

Usage profiles, i.e. usage behavior models of SIF are also considered as test-related models, since they can be used to inform and drive testing scenarios.

Several notations are used for the formulation of test-related models, from the standard UML/OCL to more domain-specific meta-models and notations.

2.2.2 On demand test automation

The potential users of the MIDAS facility are all the professionals involved in the design, execution and management of SOA testing and usage observation activities. The MIDAS facility allows businesses to start, manage and operate their own tenancy dedicated to their testing and usage observation activities.

Note that, in the most general case, a MIDAS tenancy does not own necessarily the SUTs that it tests. In this respect, a straightforward use case occurs when a tenancy wants to test APIs provided by an organization through an API management facility. A more complex case occurs when the SUT is an instantiated services architecture whose nodes are owned by different organizations. In this case, the different owners of SUT’s nodes can constitute a MIDAS community and share – totally or partially - resources such as system models, test configuration models,
test models resulting from the applications of different test schemes, usage journals and profiles and the access to the SUT for unit and integration testing.

Users will access through a GUI and/or an API, on a multi-tenant basis, front-end functionalities that can be grouped as below:

- model authoring;
- testing functionalities (test generation, scheduling, execution, arbitration, reporting) through invocation of test schemes;
- management (import, export, persistence) of data resources (such as model artifacts, test data, logs, etc.) relevant for the testing activity;
- tenancy administration – user identification, authentication and authorization - and community management;
- cloud resource allocation - accounting and billing functions.

The MIDAS facility exposes an appropriate subset of the functionalities listed above through APIs. This access modality allows seamless integration of the MIDAS functionalities with Integrated Development Environments (IDEs) and Application Lifecycle Management (ALM) platforms - for instance, the programmed invocation of automated non-regression test campaigns at specific milestones of a software engineering cycle - and, obviously, new engineering service compositions that are unforeseen at present.

The MIDAS API management facility supplies:

- a portal, for discovering, experiencing and understanding MIDAS APIs;
- a gateway for secure and mediated traffic to the MIDAS facility through the APIs.

We have already seen that the MIDAS facility proposes its available testing approaches, practices and methods as a test scheme portfolio. Firstly, the MIDAS facility shall offer a limited number of test schemes supplied by the MIDAS partners, but, in respect to test schemes, the MIDAS facility will be an open platform: it supports, in a dedicated section

---

25 The detail for each group of functionalities is an open project issue.
of the developer portal, the test scheme developer in her/his activities of design, development and packaging - in a MIDAS-compatible manner - of new test schemes to be installed on the MIDAS facility. If, after verification and evaluation, a new test scheme is added to the current portfolio, it is proposed as such to MIDAS end-users. Furthermore, MIDAS facility will propose scoring methods and contest mechanisms in order to evaluate competing test schemes (for instance, test schemes based on competing fuzzing algorithms) that are based on parameters such as the failure seeking capability.

The MIDAS facility is deployed on a cloud infrastructure allowing scalability and elasticity of hardware and software resources. The cloud deployment enhances the MIDAS facility delivery model. On one hand, test scheme developers are encouraged to design and implement testing algorithms that take advantage of the theoretically unlimited availability of hardware and software resources, for instance by putting in place distributed and parallel test generation and test execution processes. On the other hand, each test scheme should be able to evaluate from the system model, the test-related models and the user-supplied parameters that it uses to drive the test activities the amount of computing power, storage and bandwidth needed by the testing activities it drives in order to support controlled dynamic resource allocation on the cloud infrastructure.
3 GENERAL COMPATIBILITY REQUIREMENTS AND RECOMMENDATIONS

General MIDAS compatibility requirements and recommendations are independent from the application of specific test schemes. They are split into requirements and recommendations on models to be uploaded, and on the SUT deployment.

The fulfillment of the general compatibility requirements by a SUT allows basic testing of it without any need of enhanced service models and/or additional test-related models. Some basic test schemes (implementing, for instance, testing methods such as data fuzzing or elementary test practices such as crash test) exploit only the service essential model that includes the service PSMs (APIs) that are put in operation by the SUT and that must be uploaded anyway.

3.1 General requirements on models

3.1.1 Essential service platform independent model

For each target SUT, the essential service platform independent model (essential service PIM) must be uploaded on the MIDAS facility. The essential service PIM supplies essential information on the service contracts, participants (abstract systems) and services architectures that are implemented by the SUT. It is a Class model that shall be used by the SUT model (see 3.1.3 System under test model) in order to classify its elements. As a descriptive model, the essential service PIM must be accurate.

Each service PIM of a SUT must conform to the MIDAS essential service PIM meta-model, i.e. must contain all the information specified by the MIDAS essential service PIM specification, including the references to the associated Service PSM elements (see 3.1.2 Service platform specific model). It may also include additional service PIM elements that are not considered as parts of the essential service PIM and that are used by specific test schemes.

The formulation for the essential service PIM that is compatible with the MIDAS facility must be based upon the Unified Modeling Language (UML) 2.4.1 [77], extended by the Service oriented architecture Modeling Language (SoaML) 1.0.1 [63] UML Profile and Meta-model.

3.1.2 Service platform specific model

For each target SUT, the collection of service platform specific models (service PSMs - machine readable artifacts) that are referenced in the
essential service PIM must be uploaded on the MIDAS facility. From the point of view of the MIDAS facility, the service PSMs is a descriptive model that must be accurate, i.e. whose elements must be deployed on the SUT.

The formulations of the service PSMs that are compatible with the MIDAS facility must be based upon:

- WSDL 1.1/XSD [83][104][105] for SOAP 1.1 interoperability platform [64], conformant to the WS-I Basic Profile (BP) Version 1.2 [92].
- WSDL 1.1/XSD [83][104][105] for SOAP 1.2 Interoperability Platform [65][66][67], conformant to the WS-I Basic Profile (BP) Version 2.0 [93].
- WADL/XSD [80][104][105] for REST/XML interoperability platform [25][55][99][100].

The above formulation should be free of endpoint locations. If they contain any endpoint location, the service PSM can be associated with only one SUT model that provides locations that match the PSM endpoint locations.

### 3.1.3 System under test model

For each target SUT, the SUT model must be uploaded on the MIDAS facility. The SUT model is a descriptive model of the SUT topology and of the SUT locations. Each SUT model must be associated to an essential service PIM.

The SUT model describes the SUT in terms of nodes, of their ports, of the required and provided interfaces that are exposed through these ports, and of communication paths linking the ports. This description must be consistent with the associated essential service PIM, whose elements must be used as classifiers of the SUT model elements.

As a descriptive model, the SUT model must be accurate, i.e. the nodes/ports/interfaces that it describes must be actually deployed and their actual locations must equal the modeled ones.

The SUT model is a UML Deployment model. The formulation of the SUT Model that is compatible with the MIDAS facility is the Unified Modeling Language (UML) 2.4.1 [77].
3.1.4 Test configuration model

In order to invoke a test scheme on a target SUT, a user must supply to the test scheme the reference to a test configuration model (TC model) that s/he has uploaded on the MIDAS facility.

The TC model is a prescriptive model. It must be associated to a specific SUT model. The TC model specifies: (i) the SUT nodes/ports and communication paths that are test targets, i.e. targets of observations, stimuli and responses, and (ii) the connections between the SUT and the MIDAS test execution system.

The TC model is a platform independent model of the test execution system architectural configuration. As such, it specifies the skeletons of the test execution system architectural components that are deployed on the MIDAS facility at a level that is independent from any specific test execution platform. Moreover, as an architectural specification, the TC model is independent from the specific test schemes that the user can invoke on the SUT by using the TC model.

The TC model is a UML Component model. The formulation of the TC Model that is compatible with the MIDAS facility is based upon the Unified Modeling Language (UML) 2.4.1 [77] extended by the UML Testing Profile (UTP) V.1.2 [78] and by the MIDAS UTP extensions supplied as part of the MIDAS UML Profile and Meta-model.

3.1.5 Test Context

For each test scenario, i.e. the application of a test scheme to a SUT through the mediation of a TC model, the invoked test scheme must create and initialize a Test Context model element that is intended to aggregate all the information needed by the test scheme implemented behaviors to perform automated test generation, scheduling, execution and arbitration.

The Test Context must be created and initialized by the test scheme initialization behaviors in a manner that conforms to the MIDAS test scheme definition specification. The test context must be filled by the user with information supplied as arguments of the invocation of the test scheme, or later, including the references to the test configuration model - and, indirectly, to the related SUT model, the essential service PIM and the service PSM - and to test-related models.

The formulation of the Test Context that is compatible with the MIDAS facility is based upon the Unified Modeling Language (UML) 2.4.1
extended by the UML Testing Profile (UTP) V.1.2 [78] Profile and Meta-model and by the MIDAS UML Profile and Meta-model.

3.2 General requirements on the system under test

3.2.1 System under test technical interoperability platform

In order to be testable by the MIDAS facility, the system under test must be built over one or more of the technical interoperability platforms listed below:

- SOAP 1.1 interoperability platform – transmission protocol SOAP 1.1 over HTTP/1.1 [64][55] conformant to the WS-I Basic Profile (BP) Version 1.2 [92].

- SOAP 1.2 interoperability platform – transmission protocol SOAP 1.2 over HTTP/1.1 [66][67][55] conformant to the WS-I Basic Profile (BP) Version 2.0 [93].

- REST/XML interoperability platform – transmission protocol HTTP/1.1 with as Content-Type value either ‘text/xml’ or ‘application/xml’ [25][55][99][100][101].

- REST/JSON interoperability platform – transmission protocol HTTP/1.1 with as Content-Type value ‘application/json’ [25][55][36][37][101].

For the REST platforms, services must be designed following the constraint of the REST application architecture [25][24] named Hypertext As The Engine Of Application State (HATEOAS).

3.2.2 System under test initialization and recovery procedure

When exercised by the MIDAS facility, the System Under Test may crash locally or globally and/or meet local or global states that prevent the running of further tests. A partial/global initialization/recovery procedure for the SUT must be made available to the MIDAS facility.

The initialization/recovery procedure concerns the local/global SUT node setting, not the state of SUT resources involved in a specific test run of a specific test case with specific test data. The latter issue is taken into account within the recommendations for functional and security testing in the next sections.
3.3 General recommendations on the system under test

3.3.1 WS-Discovery implementation

If the SUT constitutes a multi-cast group that implements OASIS Web Services Dynamic Discovery (WS-Discovery) Version 1.1 (and consequently SOAP-over-UDP) [90][68] the endpoints of the multi-cast group could be automatically located by search based on service type and service name.

The SUT should implement WS-Discovery and SOAP-over-UDP. The fulfillment of this recommendation may allow the collection of the SUT endpoint locations that may contribute to the semi-automatic construction of the SUT Model and may consent checking that the endpoints are up.

3.3.2 WS-MetadataExchange and WS-Transfer implementation

If part or all of the located endpoints of a system under test enact reflexive capabilities through the standard Web Services Metadata Exchange [94] and Web Services Transfer [98] services, the metadata (supported WSDL and XSD) of the endpoints therein may be automatically retrieved and the services exposed by the endpoints may be checked.

The SUT should implement WS-MetadataExchange and WS-Transfer. The fulfillment of this recommendation allows the automatic retrieval of meta-data from the SUT that may contribute to the semi-automatic construction of the SUT model and may facilitate the check of the service exposition by the endpoints.

3.3.3 Service Component Architecture (SCA) deployment

Service-Component Architecture (SCA) [58] is an OASIS standard that provides a technology-independent but implementable model for developing, assembling and deploying Service-Oriented Architectures [13][14][112][2]. In particular, the components in a SCA composite application can run on different nodes in a network [60].

In SCA, the definitions of composites, components, their implementations, and the nodes they run on belong to what is called an SCA domain. SCA implementations provide administration tools that let a system administrator manage the SCA artifacts in the domain. Using the domain gives the administrator the flexibility to specify installation characteristics of nodes, such as host or port, at the time the nodes are added to the domain instead of in composite files.
The SUT should be deployed as a SCA. The use of SCA facilitates the SUT deployment and administration, independently from MIDAS testing. Moreover, the fulfillment of this recommendation may allow the semi-automatic construction of the SUT Model. The SCA deployment implementation that is compatible with MIDAS is Apache Tuscany SCA Java 2.0\textsuperscript{26}, which is compliant with the Service Component Architecture (SCA) Assembly Model Specification Version 1.0\textsuperscript{27} (SCA-Assembly).

\textsuperscript{26} http://tuscany.apache.org/home.html

\textsuperscript{27} The current draft version of the OASIS SCA Assembly Specification standard is 1.1.
4 COMPATIBILITY RECOMMENDATIONS FOR FUNCTIONAL TESTING

4.1 Functional testing

SOA functional testing is always model-based testing, i.e. testing the compliance of the service implementation with the service model. It includes two related but different tasks: (i) service interaction testing and (ii) service operation testing. Service interaction testing is performed through challenging and checking how the service parties exchange information and commitments in order to coordinate the services’ provisions/consumptions, or the denials of service provisions if all the enabling conditions are not satisfied. Service operation testing is performed through challenging and checking the service provision, i.e. what the service provider does.

Service functional testing can be performed under either service unit testing – testing the behavior of one SUT node - or service integration testing – testing the coordinated behaviors of at least two SUT nodes.

4.1.1 Service unit functional testing

We distinguish between interaction type and interaction content. Service interaction types (for instance, service operations in WSDL) can be organized by an interaction protocol that restrains the allowed exchanges of interaction types independently from their contents.

Protocol testing [5] challenges the abstract protocol implementation – it checks that the type of the actual interaction effected in a state of the actual conversation is one the interaction types allowed by the protocol in that state – but not the appropriate use of the interaction protocol in the actual functional context. Protocol testing checks: (i) that the protocol is honored in the actual interactions (protocol accuracy test) and (ii) that the SUT reacts correctly - by signaling the protocol error - when exercised with interactions that are not protocol compliant (protocol robustness test). Test generation methods such as protocol fuzzing, that we will present in the next section, can be employed within protocol testing.

Basic functional testing couples protocol and content testing. It checks that the SUT, when appropriately stimulated, responds with actual interactions that match in type and content the expected ones. Basic functional testing checks not only that the protocol is correctly implemented, but also that it is appropriately used in the specific functional context, i.e. that the interaction content is compliant with the expected one.
The most popular and effective modeling paradigm for the service functions (interaction + operation) that is exploitable for model-based testing of service functions, is the contract-based design (or Design by contract™) approach [48][28][9][16][49][43][34]. Within this approach, a service operation is formally specified by its signature and its pre/post-conditions – formulated as expressions of a formal language such as OCL [51] - that bear on the elements of the service object model formulated with a model language such as UML [77].

Basic functional testing is sufficient for full accuracy and robustness test of stateless operations. If the SUT responds to an query with an answer that is compliant with the expected one in the specific state of the test run (the state of the SUT managed resources that has been set through the initialization procedure – i.e. the balance of a bank account), we can infer that the test passed; otherwise we must acknowledge that the test did not pass (the accurate arbitration of the outcome, in terms of SUT failure or test environment error, requests additional assessment).

Basic functional testing of stateful service operations gives information about the interaction (protocol and content) but does not give any information about the change of state of the resources (e.g. a bank account balance) and the actions performed in the physical world (e.g. dispensing money by an ATM) that are involved in the service provision/consumption. The transitions of the resource states and the accomplishment of the physical actions being not checked, basic functional testing can generate:

- **false negatives**, when the actual interactions match with expected ones but either the state transitions have not been performed accurately or the physical actions have not been accomplished properly;

- **partial false positives** when the actual interactions do not match with the expected ones but the internal state transitions have been effected accurately and the physical actions have been performed properly.

The SOA principle of hiding of internals forbids the direct inspection of the implementations in order to check the resource states and the

---

28 Design by contract™ is a trademark registered by Eiffel Software in the U.S.A. (http://www.eiffel.com/).
states of the devices that perform actions in the physical world\(^\text{29}\). The stakeholder confidence in the correct handling of the resource state transitions can be increased only by cross-checking, i.e. by comparing information coming from different sources. If the SUT implements and exposes ancillary transparency services, these services can be used to retrieve the states after the service operation execution and to put in place more elaborated test cases in which the primary service interaction is entrenched with transparency services interaction, allowing the test arbitration module to cross-check the different outcomes and to deliver a test verdict supported by a more informed assessment of the test outcomes. Cross-check functional testing requires the enactment of transparency services by the SUT.

Service functional testing put in place functional accuracy test and functional robustness test. Functional accuracy test checks (through positive test cases) whether the service provision/consumption is correctly performed when all the enabling objective conditions are satisfied. Functional robustness test checks (through negative test cases) whether the service provision is correctly denied when at least one enabling condition is not satisfied (and is not performed actually).

The table below summarizes the characteristics of the unit testing practices described above, and compares them to some non-functional testing practices.

<table>
<thead>
<tr>
<th>Practice</th>
<th>Description</th>
<th>Effectiveness</th>
<th>Prerequisites on models</th>
<th>Prerequisites on the SUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Crash testing – all services</td>
<td>Looking for SUT crash</td>
<td>Crash test (non-functional)</td>
<td>(nothing)</td>
<td>(nothing)</td>
</tr>
<tr>
<td>Basic syntactic testing – all services</td>
<td>Checking well-formedness</td>
<td>Syntactic accuracy and robustness test (non-functional)</td>
<td>(nothing)</td>
<td>(nothing)</td>
</tr>
<tr>
<td>Protocol testing - all services</td>
<td>Checking the abstract interaction protocol</td>
<td>Protocol accuracy and robustness test (non-functional)</td>
<td>4.2.1</td>
<td>(nothing)</td>
</tr>
</tbody>
</table>

\(^\text{29}\) Changes of the states of the physical world that are typically non-undoable (e.g. dispensing money by an ATM) are detectable only if it is possible to retrieve the state of the device (e.g. the ATM) that produce them, and the device changes its state in a non-undoable manner when effecting the non-undoable action.
Service unit functional testing can be performed through different testing methods such as *equivalence partitioning, boundary value analysis, all-pair, state-transition tables, mutation, random*, and others, that are applied for both service unit testing and service integration testing.

### 4.1.2 Service integration functional testing

End-to-end service exchange in a services architecture *may* be explicitly modeled. If it is modeled, the model is either platform specific (at the implementable level) or platform independent (at the logical level). The implementable model of a service exchange is an *orchestration*, i.e. an *executable script* whose execution carried out by a special participant called *orchestrator* drives the service exchange. The platform independent model of a service exchange is called *choreography model* and acts as a *specification* for the service exchange. Some standards and academic approaches [83][109][17][80] have been proposed for platform specific (implementable) choreography models but their implementation have not attained the popularity of the main orchestration approach [7][8].

Testing the execution of a script and testing the compliance of an implementation with a choreography model are different tasks. The former is white-box testing of programs, while the latter is rather specification-based testing.
The currently available orchestration language for service exchange is BPEL [7][8][39][40]. The BPEL process description can be abstracted as an extended control flow diagram; paths over this diagram can be used to guide test generation and to assess test coverage. A major issue in this approach comes from the parallelism in BPEL, which results in a much more complex control flow and a very high number of paths. The BPEL process operational semantics is derived from Petri nets [33][42][32] and analyzing a BPEL process as a Petri net allows formal checking of its properties such as reachability, deadlock, starvation, .... The computational complexity of the involved model and model-checking algorithms is high, but the availability of theoretically unlimited concurrent computing resources on the MIDAS facility cloud infrastructure opens new research perspectives [9]. Research results on Petri nets, e.g. on testing coverage and reduction techniques, may be employed in testing of Web service composition based on BPEL. Namely, optimized test case may be generated based on the translated Petri net [19]. Anyway, several other techniques are available for BPEL testing [109].

The actual end-to-end service exchange that follows a stimulus can be checked for compliance with the abstract choreography - checking that the types of the actual interactions of the graph are compliant with the allowed interaction types of the choreography. As for protocol testing, choreography testing checks: (i) that the choreography and the single service protocols are honored in the actual interactions (choreography accuracy test – which is not functional accuracy) and (ii) that the constituents of the service components architecture react correctly - by signaling the choreography/protocol error - when exercised with interactions that are not choreography/protocol compliant (choreography robustness test).

Full integration functional tests can be generated only by a full control and data flow model [46][47] that specifies not only the control transition between activities, but also the formal correspondences (functions) between the data involved in ingoing and outgoing interactions for each node (transfer function). Service composition test needs the upload of the integration control and data flow model.

The table below summarizes the characteristics of the integration testing practices described above.
<table>
<thead>
<tr>
<th>Practice</th>
<th>Description</th>
<th>Effectiveness</th>
<th>Prerequisites on models</th>
<th>Prerequisites on the SUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic choreography testing</td>
<td>Checking abstract choreography</td>
<td>Choreography accuracy and robustness test (non-functional)</td>
<td>4.2.1, 4.2.3</td>
<td>(nothing)</td>
</tr>
<tr>
<td>Basic integration testing</td>
<td>Checking the control and data flow</td>
<td>Integration accuracy and robustness test</td>
<td>4.2.1, 4.2.2, 4.2.3</td>
<td>4.3.1</td>
</tr>
<tr>
<td></td>
<td>Checking the interaction protocol and content for each service component</td>
<td>Partial functional accuracy and robustness test (possible false negatives and false positives) for single service components</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Full integration testing</td>
<td>Checking the control and data flow</td>
<td>Integration accuracy and robustness test</td>
<td>4.2.1, 4.2.2, 4.2.3, 4.2.4</td>
<td>4.3.1, 4.3.2</td>
</tr>
<tr>
<td></td>
<td>Checking the interaction protocol and content and the internal state for each service component</td>
<td>Full functional accuracy, robustness, integrity and fault-tolerance for single service components</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Basic orchestration testing</td>
<td>Model checking, path analysis, control and data flow testing</td>
<td>Orchestration accuracy and robustness test</td>
<td>4.2.5</td>
<td>4.3.1</td>
</tr>
<tr>
<td></td>
<td>Checking the interaction protocol and content for each service component</td>
<td>Partial functional accuracy and robustness test (possible false negatives and false positives) for single service components</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 4.2 Recommendations on models for functional testing

#### 4.2.1 Service interaction protocol model

The service interaction protocol model is a PIM element that describes the allowed sequences of interaction types. It should be uploaded on the MIDAS facility for protocol testing.

The formulations of the service interaction protocol that are compatible with the MIDAS facility are based upon:

- UML 2 Protocol State Machine [77] - the service interaction protocol is described as a UML 2 Protocol State Machine and is nested in the SoaML Provider and Service Interface classes [63].
• UML 2 Interaction [77] – the service interaction protocol is described as a UML 2 Interaction and is nested in the SoaML Service Contract class [63].

• UML 2 Activity [77] - the service interaction protocol is described as a UML 2 Interaction and is nested in the SoaML Service Contract class [63].

The service interaction protocol should be included in the service PIM. The fulfillment of this recommendation allows protocol and functional testing.

4.2.2 Service function model

The service function model is a PIM element that describes the formal meaning of the service operation. It should be uploaded on the MIDAS facility, for basic and cross-check functional testing.

The contract-based service operation model includes four model elements: (i) the service operation signature, (ii) the service operation pre-conditions, (iii) the service operation post-conditions and (iv) the service object model.

The formulations of the service function model that are compatible with the MIDAS facility are based upon:

• UML 2 Class model [77], extended by the MIDAS UML Profile and Meta-model - for the service object model.
• UML 2 operation defined in the service contract SoaML Capability class [77][63] - for the service operation signature.
• OCL ‘pre’ and ‘post’ expressions associated to the service operation signature [77][51] - for the service operation pre/post-conditions.

The service function model should be included in the service PIM. The fulfillment of this recommendation allows basic and cross-check functional testing.

4.2.3 Service choreography model

The service choreography model is a PIM element that describes the interaction protocol between the participants of a services architecture from end-to-end service exchange perspective. The service choreography model should be uploaded on the MIDAS facility to be used for service choreography testing.
The formulations of the service choreography model that are compatible with the MIDAS facility are based upon: UML Protocol State Machine [77] the service interaction protocol is described as a UML 2 Protocol State Machine and is nested in the SoaML Provider and Service Interface class [63].

- UML 2 Interaction [77] – the service interaction protocol is described as a UML 2 Interaction and is nested in the SoaML Service Contract class [63].

- UML 2 Activity [77] - the service interaction protocol is described as a UML 2 Interaction and is nested in the SoaML Service Contract class [63].

The service choreography model should be included in the service PIM. The fulfillment of this recommendation allows basic choreography testing.

4.2.4 Service composition model

The service composition model is a PIM element that describes service composition between the participants of a services architecture. It describes in particular the formal correspondence between data involved in ingoing and outgoing interactions for each node of the SUT (transfer function).

The service composition model should be uploaded on the MIDAS facility, to be used for service composition testing.

The formulation of the service composition model that is compatible with the MIDAS facility is based upon the annotation of OCL query expressions [51] upon the choreography model (4.2.3. Service choreography model).

The service composition model should be included in the service PIM. The fulfillment of this recommendation allows service composition testing.

4.2.5 Service orchestration executable model (script)

The service orchestration script is an executable PSM element that implements the automated coordination of the service exchange in instantiated services architecture. The exchange of services therein is mediated by a service component that plays a special role called orchestrator and is equipped by an orchestration engine that runs the orchestration script.
If the services architecture under test is orchestrated, the orchestration script should be uploaded on the MIDAS facility, to be used for model-checking and automated test case generation for service orchestration testing.

The formulation of service orchestration scripts that is compatible with the MIDAS facility is BPEL4WS Version 1.1 [7] and WS-BPEL 2.0 [8].

The fulfillment of this recommendation allows service orchestration testing.

4.3 Recommendations on the system under test for functional testing

4.3.1 Resource state re/initialization service implementation

The MIDAS facility management of test cases is compliant with the ETSI specification [21] that asserts that a test run shall leave the System Under Test in the state before the test run. To comply with this rule without breaking the principle of internals hiding, the SUT should enact an ancillary service that allows the MIDAS facility to reinitialize the SUT state for the test run.

The SUT should enact a state re/initialization service on the basis of (i) the (partial) implementation of a basic generic and standard CRUD-like (create, read, update, delete) service and (ii) the implementation of a specific state re/initialization scheme for the System Under Test, i.e. of a data structure type that allows to accommodate re/initialization data and of the re/initialization software component. In this respect, the MIDAS facility is able to automatically generate the data allowing the re/initialization of the SUT state, and to invoke the re/initialization operation with the appropriate data on the SUT.

The System Under Test should put into operation the state re/initialization through the standard generic CRUD-like services:

- W3C Web Services Transfer [98];
- OMG hData RESTful Transport [31];
- OMG Retrieve, Locate, And Update Service (RLUS) [56].

The fulfillment of this recommendation allows basic functional testing.
4.3.2 Transparency service implementation

In order to run cross-check functional testing (but also for vulnerability testing – see the next section), the MIDAS facility needs to retrieve information about internal states of the System Under Test that are relevant for the service provision/consumption.

The SUT should enact a transparency (state retrieving) service on the basis of (i) the (partial) implementation of a basic generic and standard CRUD-like (create, read, update, delete) service and (ii) the implementation of a specific query scheme for the System Under Test, i.e. of a data structure type that allows to represent the state to be queried and the query software component. In this respect, the MIDAS facility is able to automatically generate and invoke the appropriate queries on the SUT state.

The SUT should put into operation the resource state transparency through the standard generic CRUD-like services:

- W3C Web Services Resource Access standards WS-Transfer and WS-Fragment [98][91];
- OMG hData RESTful Transport [31];
- OMG Retrieve, Locate, And Update Service (RLUS) [56].

The fulfillment of this recommendation allows cross-check functional testing.
5 COMPATIBILITY RECOMMENDATIONS FOR SECURITY TESTING

5.1 Security testing

One of the main differences between the traditional distributed software system engineering approach and the SOA style is loose coupling between the software components that is adopted by the latter. As a consequence, the later orchestration of different services in services architecture is not known in advance from the service providers’ points of view and less trust while interacting with other services requires a higher level of security. To achieve this level, security testing is much more important for SOA-based software than for traditional software. This is because traditional software is usually applied in a well-known or at least predictable configuration and has not to cope with its application in an unforeseeable application context or in sophisticated configurations and orchestrations as SOA-based software are faced with.

Another aspect of security testing is that more and more (in particular sensitive) data are processed by such services, e.g. by governmental institutions or in the health sector. Security issues on applications that process or store such sensitive data have a large impact on providers of such services that may be considered as legally responsible.

The MIDAS facility intends to support security testing as described in the following sections.

5.1.1 Vulnerability Testing

Security testing aims at finding faults of the SUT that may have a security impact. In general, security-related faults that are not simply the causes of security clauses compliance failure are called vulnerabilities because they allow an attacker to abuse the SUT, e.g. in order make it crash or to access data s/he is not allowed to. In order to look for such vulnerabilities, the testing perspective is moved from the system specifications to the attacker behavior. The main aspect of security testing is to stimulate the SUT with input that reveals vulnerabilities. Mostly, such inputs are invalid in the sense of the specification. Therefore, in contrast to functional testing, security testing is mostly negative testing and may be based on misuse cases instead of use cases.

In most cases, in order to look for vulnerabilities, it is not sufficient to observe the immediate response of the SUT. It is rather necessary to collect information about the internal state of the SUT in order to find
deviations in its behavior that do not become manifest in the direct response. Such deviations may be observable after sending further stimuli or by sending a certain sequence of stimuli to the SUT.

Because vendors of SOA services want that their services will be used as much as possible, the services need to be compatible with most possible systems. This is also the reason because SOA services are threatened by many different ways of attack, such as virus, malware and fault injection.

SQL Injection is frequently used to extract data from websites with backend database. Hence, it has to be guaranteed that it is not easily possible to send a special query to the database via a public website to erase them. SOA services are especially vulnerable by XML nodes and SOAP messages. Depending on the type of the database management system, SQL injection may be complemented by XQuery [103] injection. Similarly, XPath [102] can be used by an attacker to access contents of XML documents s/he is not allowed to.

### 5.1.2 Model-based fuzzing

Fuzzing is a negative testing approach where the interface of a system under test (SUT) is stressed with invalid, random or unexpected inputs [52]. It aims at finding security-relevant weaknesses in the implementation that may cause denial of service, degradation of service or undesired behavior [73] (p. 24).

The simplest fuzzers generate input data randomly and produce a large quantity of totally invalid data [73] (p. 27). Therefore, such totally invalid input data are mostly rejected by the SUT because they do not pass simple sanity checks [52]. In order to pass the sanity checks and to get invalid data being processed by the SUT, more sophisticated fuzzers have knowledge of the protocol that is used to communicate with it. Using this knowledge, they generate semi-valid input data, which means that such data is valid in most parts but invalid in a few [52]. Such data have a better chance to pass possibly faulty sanity checks of the SUT and to be processed by the SUT. Thus, semi-valid data may reveal security-relevant weaknesses in the SUT’s implementation [27][108][26][79].

Model-based or smart fuzzers have a complete model of the SUT’s protocol [74] and are able to generate complex interaction with it by sending a set of valid messages and after reaching a certain state, stressing it with semi-valid input data. Therefore, while random fuzzers often find just simple bugs, more complex bugs can be found by model-
based fuzzers [73] (p. 28). Another consequence is that model-based fuzzers are more efficient while the targeted input space is much smaller than the input space of all invalid inputs targeted by random-based fuzzers as depicted in Figure 1.

![Figure 1: Targets of Random-Based and Model-Based Fuzzers (cf. [73], p. 20)](image)

In functional testing, the test verdict is determined by comparing the expected and the actual responses received from the SUT when stimulating it with specified inputs. Test verdict arbitration is much more difficult while fuzzing because it aims at finding deviations in the SUT’s behavior. A crash or a hang of the SUT can be easily detected by a connectivity check [72] (p. 472) or valid case instrumentation where a functional test case is executed after each fuzz test case to determine if the SUT is still functional [73] (p. 170).

Measuring the testing process in order to determine if the already executed test cases are sufficient is crucial. However, this is a topic where only little research results are available. There are some metrics for measuring the fuzzing process. While some are based on coverage of the specification or the injected anomalies in order to generate invalid or semi-valid input data, others refer to code coverage of the interfaces (attack surface) [73](pp. 120-133). Microsoft made an empirical investigation on how much test cases are necessary to cover most of the bugs based on Windows Vista file parsers for different file formats and recommend executing at least 500,000 test cases, and using another fuzzing tool when no bugs were found since 250,000 test cases) [61]. But it seems questionable if these results are valid for other than the used tools and SUTs.

Fuzzing approaches may be categorized in data fuzzing and behavioral fuzzing. With data fuzzing the SUT is exercised with invalid input data while behavioral fuzzing consists of submitting invalid message sequences to the SUT.
Web services offer a well-defined interface that is formally described, e.g. by using Web Service Description Language (WSDL). Such a specification can be used as a basis for data fuzzing by working included data descriptions in order to generate invalid input data. For that purpose, the data type specification as well as messages that are exchanged with a service component can be used to create a set of stimuli where some of them carry invalid input data. For test data generation, different approaches are possible, e.g. using the web service description as a collection of constraints on input data and modify these constraints for generating invalid input data.

Web service descriptions are also a good basis for behavioral fuzzing. In contrast to data fuzzing, invalid messages sequences are the focus of behavioral fuzzing. WSDL provides the messages that can be exchanged with the service. However, this is not sufficient for behavioral fuzzing. Additional to the messages provided by a service, we need the correct order of messages in order to generate invalid message sequences. Such a specification of the flow of messages may be provided by security policies (e.g. WS-SecurityPolicy [97]) that may constitute a basis for generating behavioral fuzz test cases.

In respect to SOAP-based web services, Representational State Transfer Interfaces (REST) as well as JavaScript Object Notation (JSON) are getting more popularity. Data exchange via JSON is, in contrast to XML documents, less restrictive because of the lack of a document format description that can be enforced. Whereas XSD documents describe with a degree of precision the data format of a valid XML document that can be automatically checked by an XML parser, such a specification is missing for JSON data exchange and allows to submit invalid input data that has to be detected by a web services that receives such data. Therefore, input validation is much more difficult and constitutes a security risk in contrast to XML that is increased by the fact that JSON documents may contain JavaScript code that may allow code injection. Hence, data fuzzing of JSON documents may also be relevant for security testing of web services using this data format.

5.2 Recommendations on models for security testing

5.2.1 Service interaction protocol model

The service interaction protocol model is a PIM element that describes the allowed sequences of interaction types. It should be uploaded on the MIDAS facility for security testing.
The formulation of service interaction protocol that is compatible with the MIDAS facility is UML Interaction [77]. The service interaction protocol should be included in the service PIM.

The fulfillment of this recommendation allows vulnerability testing.

5.2.2 Security test model

The security test model is a model of the objectives and plans of security testing. It should be uploaded on the MIDAS facility for security testing.

The formulation of the security test model that is compatible with the MIDAS facility is UML Testing Profile (UTP) Version 1.2 [78].

The fulfillment of this recommendation allows security testing.

5.2.3 Security policy machine readable model

The security policy machine readable model is a PSM element that describes the policies on service confidentiality and authenticity exigencies and constraints. It should be uploaded on the MIDAS facility for security testing.

The formulation of the security policy machine readable model that is compatible with the MIDAS facility is WS-SecurityPolicy 1.2 [97].

The fulfillment of this recommendation allows policy-based security testing.

5.2.4 Encryption/decryption and signature/validation algorithms and keys

In order to check the compliance of encryption/decryption, signature/validation implemented procedures, the MIDAS facility needs access to information such as security keys and related algorithms.

The encryption/decryption and signature/validation algorithms and keys should be uploaded on the MIDAS facility for security testing. The format of the aforementioned information that is supported by the MIDAS prototype is still an open issue.

The fulfillment of this recommendation allows policy-based security testing. The fulfillment of recommendation 5.2.3 is a prerequisite for this one.

5.3 Recommendations on the system under test for security testing

5.3.1 Resource state re-initialization service implementation
See 4.3.1 (Resource state re-initialization service implementation).

The fulfillment of this recommendation allows *security testing*.

5.3.2 Transparency service implementation

See 4.3.2 (Transparency service implementation).

The fulfillment of this recommendation allows *vulnerability testing*. 
6 Compatibility Recommendations for Usage-based Testing

6.1 Usage-based testing

SOA usage-based testing is a new promising research and engineering approach about SOA testing. Usage-based testing is a technique that generates and/or chooses test cases such that the testing effort is based on usage-based considerations.

The MIDAS facility will provide the means for usage-based testing (see, e.g. [82][75][76][57][70][71][29][30]) of service architectures. A popular approach focuses on the highly used paths of interaction within the SUT. The intent is to focus on highly used functions of the service and highly usual end-to-end service exchanges in services architectures. Usage-based testing is a technique that generates test cases such that the testing effort focuses on the highly used parts of the SUT. As a by-product of this approach, it is possible to determine the reliability (the rate of failures given a usage scenario) of the SUT with respect to the current usage and utilize this measure as a criterion within acceptance testing. Conversely, focusing on lowly used paths of interaction within the SUT is interesting for augmenting the functional test coverage.

Usage-based testing is intended to support the functional testing activities of the MIDAS facility. Functional testing methods such as systematic or random testing will test all parts of the SUT equally, which ensures a certain level of testing (depending on the functional testing method employed) for the SUT as a whole. In combination with usage-based testing, the highly used parts of the SUT are tested intensively, over the level of the overall SUT.

Usage-based testing is grounded on the concept of usage profiles. The usage profile describes the usage of a system in stochastic terms, e.g., the probability of the next action of a user or the next service operation invocation. The usage profiles will be used to generate test cases, evaluate the usage-based coverage of test suites and schedule test cases prioritized by the usage they cover. The MIDAS facility provides a complete usage-based testing process, from the usage profile inference over the test case generation to the evaluation of usage-based tests and their integration with functional testing.

Usage profile models are built from information extracted from usage journals that are issued from usage observation on the SIF. The MIDAS facility shall support the user in producing easy-to-put-in-place
mechanisms (observer software) that are able to perform usage observation. The observer software is placed by the SIF administrator on the chosen SIF usage observation points and it provides MIDAS-compatible usage journals. The observer software is able to capture, store, and retransmit without any change and with a minimal overhead the interactions between the nodes at the communication path endpoints.

The MIDAS usage observation facility supports end-users in two activities: (i) building and downloading the observer software that will be installed on the SIF at the usage observation points and (ii) collecting and uploading the usage journals produced by these observer software components installed on the SIF.

The SIF and the SUT are physically different systems that are usually managed by users that are employed by different organization (for instance, respectively the operation and the engineering departments).

The system models (essential service PIM, service PSM, SUT/SIF model) of the SUT and of the SIF shall conform to the same meta-model\textsuperscript{30}. Moreover, the SUT for usage-based testing and the associated observed SIF shall share the essential service PIM and the service PSM. Roughly speaking, they must exhibit the same services architecture. Furthermore, the SUT for usage-based testing shall be isomorphic to the associated observed SIF or to a subsystem of the observed SIF\textsuperscript{31}.

\section*{6.2 Recommendations on models for usage-based testing}

\subsection*{6.2.1 Usage journal for usage profile inference}

The usage journals of the SIF are needed by the MIDAS facility for usage profile inference. The SIF usage journal should be uploaded on the MIDAS facility. The upload of the SIF usage journal is a prerequisite for usage profile inference.

The format of the usage journal that is supported by the MIDAS facility is specified by the MIDAS usage journal specifications and documented on the MIDAS platform.

The fulfillment of this recommendation allows usage profile inference for usage based testing.

\footnote{The only difference is that the SUT model Nodes are located (their location property must be fulfilled) while the SIF model Nodes are not located (only the SIF topology is relevant).}

\footnote{More precisely, their accurate models shall be isomorphic.}
6.2.2 Recommended models for functional testing

Usage-based testing is intended to support functional testing on the MIDAS facility.

In order to allow usage-based support of different practices of functional testing, the MIDAS facility should ensure the interoperability of the usage-based test models and the models needed for functional testing.

The fulfillment of the compatibility recommendations expressed in 4.2 (Recommendations on models for functional testing) is a prerequisite for usage-based functional testing.

The fulfillment of this recommendation allows usage-based support for functional testing.

6.3 Recommendations on the system in the field for usage-based testing

This section expresses the compatibility recommendations that a system in the field (SIF) should adopt in order to supply information exploited by the automated usage profile inference of the MIDAS facility.

6.3.1 System in the field model compliance

The models describing the SIF shall be the same as the models describing the SUT with the exception of the location property of Nodes/ports.

The fulfillment of this recommendation allows usage-based testing.

6.3.2 Usage data observation

The SIF shall enact usage observation mechanisms to collect usage information during its execution. The information should be gathered in a usage journal.

The SIF should use the usage observation components supplied by the MIDAS facility in order to collect usage information and fill the usage journal. The use by the SIF of the observers supplied by the MIDAS facility ensures full compatibility of the SIF usage journal.

The usage observers to be downloaded from the MIDAS facility are not to be intended as observer nodes hosted on the MIDAS facility but as software components that make easier the implementation on the SIF of the usage journal.

The fulfillment of this recommendation allows usage profile inference and consequently usage-based testing.
6.3.3 Usage data accuracy

The SIF should enact usage data observation by fulfilling the recommendation 6.3.2 (Usage data observation) in true continuous field conditions, in order to guarantee that the usage journal registers accurately the actual usage and that the MIDAS facility infers a sound usage profile.

The fulfillment of this recommendation allows sound usage profile inference and consequently improved usage-based testing.

6.4 Recommendation on the system under test for usage-based testing

6.4.1 Recommended enactments for functional testing

Usage-based testing is intended to support the functional testing activities of the MIDAS facility.

In order to allow usage-based support of different practices of functional testing, the fulfillment of the compatibility recommendations expressed in 4.3 (Recommendations on the system under test for functional testing) is a prerequisite.

The adoption of this recommendation allows usage-based support for functional testing.
7 ANNEXES
7.1 Service architecture for Healthcare – pilot general description

7.1.1 The background (EHR, integration among care settings, continuity of care...)

The Healthcare sector represents a critical as well as vital domain of the "second economy"[10], since each clinical and/or administrative act is the result of complex processes involving several heterogeneous actors. Despite these features, in the past, the evolution of healthcare information systems had typically a bottom-up approach; starting from the specific needs had single departments or units, specific point to point integrations were developed in order to guarantee interoperability among them. But, care and assistance needs are today changing as a consequence of social, economic and technological transformations occurred in the last decade. Healthcare spending has strongly been increased in all industrial countries and, in Europe, it will potentially climb up to about 15% of the GDP (Gross Domestic Product) by 2020. This growth is largely driven by the increase in demand due to the average life expectation and, as consequence, to the number of people affected by chronic diseases which become more expressed with age progress. This new situation has three different but interconnected perspectives:

1. the managers of the clinical services are challenged to deliver effective health care services at reasonable costs;

2. the clinicians are interested to apply and improve the effectiveness of clinical guidelines, increasing the emphasis on evidence-based medicine;

3. the citizens are increasingly demanding high quality and continuity of care, they are more aware of clinical risk management and are claiming more efficient healthcare services, with reduction of waiting times and a better utilization of resources.

Innovative solutions, implementing regional and national Electronic Health Record (EHR) and supporting the integration of traditional care/assistance procedures, centered around hospitals, with new forms of assistance based on telemedicine and remote monitoring at home or in peripheral structures, are promising to foster the continuity of care and the realization of integrated assistance processes. For this regard, a key role is played by:

- the national initiatives promoting the implementation of EHRs,
• the European initiatives promoting the definition and the adoption of healthcare standard interoperability technologies and services (EPSOS project\textsuperscript{32}) and,

• the research activities for the realization of Personal Health Systems supported by the European “ICT for Health” DG INFSO Unit in the VI and VII Framework Programme.

These solutions could allow healthcare government policy makers and services providers to guarantee in the next years quality and continuity of care in a sustainable way.

In this context, important industrial players are investing significant resources to develop innovative solutions for the implementation of really interoperable EHRs. The first generation of EHRs focused on interoperability among the actors of a healthcare information system, by implementing integration profiles promoted by standardization initiatives such as IHE\textsuperscript{33} and by adopting communication and data representation standards such as HL7\textsuperscript{34}, CDA\textsuperscript{2} and DICOM\textsuperscript{36}.

It is now emerging the need of cooperation among different healthcare affinity domains, both at regional/national level and in cross-border scenarios. Moreover, in a spreading digital economy, where the citizen is able to organize travels, buy and sell without intermediaries, and evaluate, judge and guide the choices of many types of services, healthcare cannot continue to be a service where the user is passive.

This trend is reflected in EHR infrastructures which are becoming increasingly arranged and enriched so that the citizen/patient can be more involved in her/his own care, gradually moving away from a passive role. At the same time, for guaranteeing continuity of care across different care

\textsuperscript{32} epSOS – the European eHealth Project, http://www.epsos.eu/

\textsuperscript{33} Integrating the Healthcare Enterprise (IHE). IHE is an initiative by healthcare professionals and industry to improve the way computer systems in healthcare share information. http://www.ihe.net/

\textsuperscript{34} Health Level Seven International (HL7) is the global authority on standards for interoperability of health information technology - http://www.hl7.org/


\textsuperscript{36} Digital Imaging and COmmunications in Medicine (DICOM) - http://medical.nema.org/
settings it is necessary to provide healthcare operators with an integrated set of services and facilities for defining, monitoring, evaluating and fine-tuning personalized care programs.

To make this transition technically and economically viable, it is necessary to consider a definitive move towards new organizational models able to govern seamless integration across government, community and private information systems (EHR and PHR systems, Telecare centers ...). Such models must be supported by a distributed architecture in which users, systems, applications and devices are able to collaborate. In addition, new delivery (and business) models need to be designed that allow flexible, continuous and sustainable services.

Service oriented architectures (SOAs), based upon largely accepted standards, could allow heterogeneous systems for effectively supporting the integration of different care environments and the integration of personal, social and environmental data that affect the citizen/patient’s healthcare processes. The actual introduction of the SOA approach could then progressively push the healthcare sector towards a technologically sound and interoperable ecosystem based on a shared vision and the cooperation between the stakeholders. On the other hand this approach needs the introduction and the application of methodologies and technologies for addressing all the functional and non-functional reliability (accuracy, robustness, availability, integrity, fault tolerance and active and passive testability) and security (authentication, authorization, confidentiality, accountability, non-repudiation).

In the next section (7.1.2) we briefly describe the interoperability platform promoted by Dedalus, in order to support regional EHR systems. In section 7.1.3 we then describe the Healthcare system used in the Healthcare pilot of the MIDAS project. It is a prototypal evolution of the Dedalus solution towards a fully SOA and model driven approach, with the adoption of standard services defined in the HSSP project\(^{37}\) and being under implementation within the context of the Italian R&D HealthSOAF project\(^{38}\).

\(^{37}\) Healthcare Services Specification Program (HSSP) is a collaborative effort between HL7 and the OMG to identify and document service specifications, functionality, and conformance supportive and relevant to healthcare IT stakeholders and resulting in real-world implementations. http://hssp.wikispaces.com/

\(^{38}\) HealthSOAF project aims to implement a plug-and-play service architecture framework of SOA services, based on HSSP international standards. http://healthsoaf.it/SitePages/hshome_en.aspx
7.1.2 The interoperability platform of Dedalus: X1.V1

The interoperability platform X1.V1 has been defined and developed in the framework of the project “Rete MMG”\(^{39}\) promoted and funded by the Region of Abruzzi and the Italian Ministry of Health, with the objective to realize a regional EHR for integrating the software of the GPs with the secondary care information systems and for supporting integrated care processes of patients. X1.V1 is fully compliant with the Italian guidelines for EHR and EPR systems (provided in the official government bulletin in March 2009).

The X1.V1 architecture includes the following modules:

- the directory of the operators, for the management of the accesses. This module interacts with a Single Sign On component, in order to support access control of multiple related, but independent software systems;
- the Master Patient Index, for maintaining consistent, accurate and current demographic and essential medical data on the patients;
- The Document Repositories, both distributed in the local healthcare information systems and centralized in the X1.V1 platform, for storing the clinical documents of patients;
- The Document Registry, for managing the index and the references of all documents stored in the repositories;
- The event managers, for notifying the actors of the presence of new documents and the occurrence of new healthcare events;
- An access gateway integrated with an Enterprise service bus (ESB).

External modules interacting with X1.V1 are:

- external document source modules, that produce clinical and healthcare documents for the HER, and
- external document consumer modules need to access documents stored in the EHR systems.

The healthcare actors involved in the processes supported by X1.V1 are:

- General practitioners,

\(^{39}\) Network of General Practitioners.
• Operators of the local healthcare authorities,
• Specialists,
• Citizens,
• Government authorities (Ministry of Economy...),
• Others.

7.1.3 X1.V1 and the interoperability issues

The management of a healthcare information system involves healthcare professionals, administrative people, and citizens, requires the application of complex processes, and needs the integration between different kinds of information. For this reason, the effort for addressing technological and semantic standardization in the eHealth arises from some years ago. In particular, we can identify three phases:

1. In mid-nineties, international organizations, such as HL7, focused on the standardization of messages that different actors had to exchange for sharing clinical and healthcare information. Moreover, the standardization process focused on the adoption of shared documents structure, by means of the HL7 CDA.

2. Since year 2000, another international organization, IHE, focused its attention on the integration profiles, with the definition of use cases. In particular, IHE Integration Profiles describe a clinical information need or workflow scenario, and document how to use established standards (e.g. HL7, DICOM, LOINC40...) to accomplish it. A group of systems that implement the same Integration Profile address the need/scenario in a mutually compatible way. Each IHE integration profile defines the actors involved in the use cases and the transactions needed for implementing the use cases. The actors and the transactions of the IHE integration profiles are specified in detailed documents, named IHE Technical Frameworks.

3. Since year 2005, in order to fully support the realization and an actual evolution of the EHR systems, the stakeholders of the eHealth market perceive the need to move the standardization process towards the SOA. To face these emerging issues, an innovative challenge in SOA research and industrial development is

40 Logical Observation Identifiers Names and Codes (LOINC®) is a universal code system for identifying laboratory and clinical observations. http://loinc.org/
represented by the Healthcare Services Specification Project (HSSP),
a joint initiative between HL7 and OMG, whose aim is to specify, by
means of a model-driven approach, a number of generic services for
the healthcare sector. For each HSSP service, HL7 produces a
standardized Service Functional Model, at the Platform
Independent Model level, and the OMG produces a compliant
Service Technical Model, at the Platform Specific Model level,
accompanied by machine-consumable elements (XSD, WSDL, and
WADL). In particular, HSSP follows the SOA approach for business
transformation based on dividing complex environments into well
defined, formally specified functions based on the activities they
perform (services). Each service has well defined responsibilities
and authority. These services are able to work together within
business processes, and can eventually be orchestrated in order to
manage their coordination and performance. HSSP introduces the
good practice of loose coupling between services behavior and
semantic contents (Semantic Signifier in HSSP jargon), with the
separation between services behavior specifications and Semantic
Signifiers Implementation Guides (IGs). Semantic Signifiers will be
based on HL7 RIM\(^{41}\) and data types.

The first generation of the Dedalus interoperability platform X1.V1
includes the first two features of the standardization process. In particular,
X1.V1:

1. adopts the HL7 (v2.x\(^{42}\) and v3\(^{43}\)) standards for the documents
   architecture and the data communication;

---

\(^{41}\) HL7 Reference Information Model (RIM). RIM is a large, pictorial representation of the HL7 clinical
data (domains) and identifies the life cycle that a message or groups of related messages will carry. It is
a shared model between all domains and, as such, is the model from which all domains create their
messages. The RIM is an ANSI approved standard. http://www.hl7.org/implement/standards/rim.cfm

\(^{42}\) The HL7 version 2 standard (also known as Pipehat) has the aim to support hospital workflows. It was
originally created in 1989. It defines a series of electronic messages to support administrative, logistical,
financial as well as clinical processes. The last version is v2.7. The v2.x standards are backward
compatible (e.g., a message based on version 2.3 will be understood by an application that supports

\(^{43}\) The HL7 version 3 standard aims to support all healthcare workflows. Development of version 3
started around 1995, resulting in an initial standard publication in 2005. The v3 standard, as opposed to
version 2, is based on a formal methodology (the HDF) and object-oriented principles. -
2. is compliant with the most important IHE integration profiles for supporting the EHR systems and the interoperability among different health environments.

Regarding to the documents structure, X1.V1 manages the following kinds of documents and metadata according to the standard HL7/CDA2. CDA is part of the HL7 v3 standard; it is based on the HL7 Reference Information Model (RIM). The CDA defines a multi-level architecture where each level is derived from a more basic level. The Level One consists of three technical specifications: CDA Header, CDA Body and HL7 Datatypes. The CDA specifies that the content of the document consists of a mandatory textual part (which ensures human interpretation of the document contents) and optional structured parts (for software processing). The currently defined and managed (CDA2) documents are:

- **ePrescriptions.** The ePrescription is an electronic health document, digitally signed and archived into the EHR. It is generated by the GP/Specialists (originator) that performs the prescription, and is used by the pharmacists, hospital personnel or specialists (recipient) to deliver drugs/equipments, to book and perform the visit, or to hospitalize the patient. The document can contain the following possible prescriptions:
  - pharmaceutical,
  - for (specialist) visit or (diagnostic) examination,
  - rehabilitation,
  - admission request,
  - medical equipment or product, and
  - request of transportation.

- **Medical reports:**
  - laboratory,
  - radiology,
  - first aid, and
  - generic (outpatient, discharge letter, etc.)

- **Patient Summary.** The Patient Summary (PS) is an electronic health document, digitally signed and archived into the EHR. The aim of the PS is to take a snapshot of the clinical history of the patient. It is updated by the GP, whenever a relevant change occurs in the health
status of the patient. The relevance of the information is established directly by the GP, being part of her/his professional responsibility. It includes a predefined set of data to be used in emergency scenarios (emergency data set). It may include links to medical reports the GP assumes of any use for the comprehension of the clinical picture of the patient. It cannot be created automatically, starting from information archived in the EHR.

- The Patient Care Coordination Report (PCCR). It is an electronic health document, digitally signed and stored in the EHR. The PCCR is composed of the overall clinical information of the patient, as collected by the GP. It does not only include that information relevant to assess/evaluate the current state of patient health, but it actually represents his or her health history (including closed problems). The PCCR can be accessed by: the patient, the GP who created it, groups or networks of associated GPs. It is useful whenever a patient’s health history must be communicated among different health professionals: e.g. as the patient revokes the mandate/trust to her/his family doctor in order to choose a new one.

Concerning the compliance with the IHE integration profiles, X1.V1 supports the following data flows:

- **Demographic integration**: Integration of the users’ demographic details. X1.V1 implements the IHE PIX (Patient Identifier Cross-Referencing) and PDQ (Patient Demographics Query) integration profiles. PIX supports the cross-referencing of patient identifiers from multiple Patient Identifier Domains by transmitting patient identity information from an identity source to the Patient Identifier Cross-reference Manager and by providing the ability to access the list(s) of cross-referenced patient identifiers either via a query/response or via an update notification. PDQ lets applications query a central patient information server and retrieve a patient’s demographic and visit information.

- **Documents sharing**. X1.V1 implements the actors Document Registry and Document Repository of the IHE XDS* (Cross-Enterprise Document Sharing) integration profiles. The XDS profile enables a number of healthcare delivery organizations, belonging to a clinical Affinity Domain, to cooperate in the care of a patient by sharing clinical records in the form of a document as they proceed with
their patients' care delivery activities. The profile includes four actors: the Document Repository which is responsible for storing documents in a transparent, secure, reliable and persistent way and for responding to document retrieval requests. The Document Registry is responsible for storing information about those documents so that the documents of interest for the care of a patient may be easily found, selected and retrieved independently of the repository where they are actually stored. Document Sources are responsible to produce and publish both documents and metadata. Document Consumers should be able to query a registry and retrieve documents. X1.V1 is then able to interact with the systems that implement document source and consumer actors. In particular, X1.V1 allows the XDS based management of documents described above.

- **Event management**: X1.V1 is compliant with the IHE NAV (Notification of Document Availability) profile. NAV defines a mechanism for point-to-point notifications between systems or users within an XDS Affinity Domain. These notifications can be used to trigger various activities within clinical and healthcare processes.

- **Authentication and authorization process.** An Actor needs to be authenticated in order to access the X1.V1 platform. Authorized users are granted with access rights compliant with their roles and capabilities. X1.V1 uses the standard SAML\(^\text{44}\) for exchanging authentication and authorization data between security domains.

  X1.V1. exploits these data flows for supporting clinical processes that allow continuity of care, integration between care settings and new policies of clinical governance aiming at improving the effectiveness and the efficacy of care. The most important processes are:

  - ePrescription,
  - diagnostic and therapeutic pathways, and
  - ebooking.

\(^{44}\) Security Assertion Markup Language (SAML) is an XML-based open standard data format for exchanging authentication and authorization data between parties, in particular, between an identity provider and a service provider. SAML is a product of the OASIS Security Services Technical Committee. https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security
7.1.4 Towards a SOA approach: the implementation of the HSSP services

The healthcare information systems require more and more interactions among heterogeneous actors, interoperability among different care settings and integration with different domain applications, such as social, environmental, administrative and so on.

To guarantee actual interoperability, efficient integration and functional autonomy, the healthcare information systems have then to be able to manage loose coupling interactions among each other.

The SOA approach is the natural answer to these requirements and an innovative challenge in SOA research and industrial development is represented by the HSSP project (a joint initiative between HL7 and OMG, whose aim is to specify, by means of a model-driven approach, a number of generic services for the healthcare sector).

Dedalus is involved in a research project (HealthSOAF), partially funded by the Italian Ministry of University and Research, whose objective is to implement a HSSP-compliant service framework based on the X1.V1 platform. Starting from the X1.V1 solution, Dedalus will design and implement the following HSSP services and the enabled service provider components:

- Health record management service (HSSP RLUS - Read, Locate and Update Service), for handling distributed patient records and data (profile, medications, clinical history, ...).
- Demographic and identity management service (HSSP IXS - Identity Cross-Reference Service), for uniquely identifying various kinds of entities (e.g. people, patients, providers, devices and so on) within disparate systems within a single enterprise and/or across a set of collaborating enterprises.
- Terminology management service (HSSP CTS2 - Common Terminology Service), for managing clinical and healthcare terminology and coding.
- Decision support service (HSSP DSS - Decision Support Service), for searching, inquiring and executing healthcare decision-aid modules.

The objective is to provide the following service bricks for each HSSP service:

- *Service consumer proxy library*, allowing applications, systems, intelligent devices to access service providers;
• **Service provider skeleton library**, allowing systems endowed with enabling capabilities to provide the service;

• **Service provider system** that is a full implementation of the service provider.

For the design and the development of these services, Dedalus is following a contract-based, model-driven service orientated approach. In particular, services are described by a set of multi-layered models:

• Service Computation Independent Model (CIM),

• Service Platform Independent Model (PIM),

• Service Interoperability Platform Specific Model (Interoperability PSM)

These service models are collections of functional (and non-functional) requirements for implementations.

The HSSP OMG specifications are models at the PIM / Interoperability PSM levels of the HSSP generic services. HL7 is in charge of providing the service functional models of the HSSP services.

Before implementing the services described above, Dedalus and the other partners of the HealthSOAF project did a reverse formal modeling of the OMG specifications at the PIM level, by checking the compliance with the corresponding HL7 Service Functional Models.

The research strategy of Dedalus is to implement the HSSP services as new functionalities and interfaces of its interoperability platform X1.V1. In particular, Dedalus is developing service provider skeletons for encapsulating a IHE compliant MPI (Master Patient Index) in order to provide IXS service functions and for encapsulating an IHE XDS registry/repository in order to provide RLUS service functions. This means that the X1.V1 will be the service provider system, with the objective to use the HSSP standard interfaces both for:

• the interaction with other enterprise level interoperability platforms (see Figure 2)

• the interaction with other Cross Enterprise level (Regional, national ...) interoperability platforms (see Figure 3)
Starting from these scenarios, the healthcare Pilot of the MIDAS project will be a service oriented architecture where some healthcare software modules will interact with the X1.V1 platform through the HSSP interfaces.
7.2 Service architecture for Supply Chain Management – pilot general description

7.2.1 The background: Supply Chain Management in Context

The Council of Logistics Management\(^{45}\) defines logistics as "the process of planning, implementing and controlling the efficient flow of goods and information related to them, from the point of origin to point of consumption in order to meet customer requirements ".

The globalization of markets and the capillarization of distribution networks present a growing challenge for companies, who have to find solutions to improve communication between the agents involved in global supply chains and make choices that benefit networks overall business.

The development and application of new technologies facilitate the development of mechanisms to carry out planning and organization of operations more efficiently. The ICT solutions are efficient tools to support management and bring value and efficiency to the internal and external processes, due to their architecture, the solution presented set all the needed information and let consult it in real time.

The Supply Chain Operations Reference (SCOR\(^{\circledast}\))\(^{46}\) model is the product of Supply Chain Council (SCC). The SCOR model captures the Council’s consensus view of supply chain management. While much of the underlying content of the model has been used by practitioners for many years, the SCOR model provides a unique framework that links business process, metrics, best practices and technology features into a unified structure to support communication among supply chain partners and to improve the effectiveness of supply chain management and related supply chain improvement activities. Member companies pay a modest annual fee to support Council activities. All who use the SCOR model are asked to acknowledge SCC in all documents describing or depicting the SCOR model and its use.

\(^{45}\) Supply Chain Council (SCC) is a global nonprofit organization whose framework, improvement methodology, training, certification and benchmarking tools help member organizations make dramatic, rapid, and sustainable improvements in supply chain performance. http://supply-chain.org

\(^{46}\) http://supply-chain.org/
SCOR® is typically used to identify, measure, reorganize, and improve supply chain processes. This is accomplished by a cyclic process of:

- Capturing the configuration of a supply chain. A supply chain configuration is driven by:
  - Levels of aggregation and information sources;
  - Source locations and products;
  - Production sites and methods;
  - Delivery channels, inventory deployment and products;
  - Return locations and methods;
  - Management of performance, data, assets and networks.

- Measuring the performance of the supply chain and comparing against internal and external industry goals. Supply chain performance is focused on:
  - Reliability - achievement of customer demand fulfillment on-time, complete, without damage etc.;
  - Responsiveness - the time it takes to react to and fulfill customer demand;
  - Agility - the ability of supply chain to cope with elastic demand within a given planned period;
  - Cost - objective assessment of all components of supply chain cost;
  - Assets - the assessment of all resources used to fulfill customer demand.

- Re-aligning supply chain processes and best practices to fulfill unachieved or changed business objectives.

To analyze the trend in supply chains, by taking into account the latest Global Supply Chain Survey 2013⁴⁷, next-generation of supply chains will be efficient, fast and tailored. Maximizing supply chain flexibility and managing multiple supply chain configurations have become the new imperatives for today’s supply chain executives. In addition, radio-

---
⁴⁷ www.pwc.com/GlobalSupplyChainSurvey2013
frequency identification (RFID) and other digital technologies lead to new frontiers in supply chain transparency and process automation. Those technologies enable multiple supply chain partners along the value chain to seamlessly interact in the joint design, manufacture, delivery and service of complex customer orders.

7.2.2 The interoperability platform of Aragon Institute of Technology: ITChain (Information Technologies for Collaborative Supply Chain)

ITChain is an infrastructure for cataloguing, publishing, discovering, performing, and composing services dynamically mobility and context-dependent, to facilitate interoperability of existing systems in a supply chain and to provide the various actors that compose global information to make decisions collaboratively.

From the industrial point of view, this platform will be implemented in three stages within the field of logistics, to facilitate decision making at all stages of procurement, distribution and transportation, so that the integration of information services should allow in the temporal axis:

- Forecast: aimed at joint planning;
- Coordination with implications: for taking decisions in real time;
- Monitoring: continuous improvement aimed at learning from experience.

Currently, the supply chain is deficient in a global and real time vision as well as an overview of all the processes that constitute the chain. This in turn causes logistical inefficiencies such as superabundance of stocks in the intermediate agents, stock-outs, high energy costs and low traceability among others. Besides, the decision-making process is ever more complex due to the uncertainty of demand and lack of information. Therefore a set of global information is shared with the agents of the supply chain to support the decision-making processes.

This platform currently covers the informal definition of business processes and their execution following a workflow engine. Information is shared between supply chain systems by using GS1 (see below).
The ITChain’s main features are:

- It is an infrastructure that enables dynamic composition of services, thus integrating different computer systems at runtime. This requires a cataloging semantics, a distributed repository, a search based on that entry, and execution and composition to ensure the safety, identity, and distributed transactions.

- The platform allows access to services on mobile devices, adapting the interface for the device and the existing connectivity at all times.

- The services are dependent on the context, i.e. the discovery, execution and their composition is not comprehensive, but will be determined by the local targets that each officer of the chain is, thus changing the services found, the composition made or the information provided at all times.

- The platform considers as a service to both sensors and actuators, wireless sensor networks form, can be discovered and implemented depending on the context to facilitate automatic identification of people and goods.

- The infrastructure allows a module to execute workflows, representing the processes of the sector as discrete events, and facilitates the traceability between the decisions.
7.2.3 ITCchain and the SOA & interoperability issues

In recent years the use of IT in logistics has become a key element to improve the competitiveness of enterprises and the adoption of SOA has emerged strongly in this area. It is worth noting the following reviews:

- “a few overriding themes emerge: collaboration, IT capability modernization, and implementation clock speed”\(^{48}\).

- “Cloud-based technology provides one place where managers can go to monitor critical supply-chain events. When an element is updated, everybody gets the information. That element of sharing is what distinguishes the cloud from more traditional means of centralizing data. All relevant data is held in one place”\(^{49}\).

- “Cloud services can also help organizations gain greater visibility, control and coordination, because they are forced to move from internally oriented sequential supply chains to interconnected and interdependent demand-driven value networks”\(^{50}\).

- “Although the use of cloud services to manage manufacturing operations is currently limited, viable options exist for some functions, particularly quality management; production planning and scheduling; environmental, health and safety; and operations intelligence”\(^{51}\).

ITChain platform uses the GS1 standard to interchange information between systems. GS1\(^{52}\) is an international not-for-profit association with Member Organizations in over 100 countries. GS1 is dedicated to the design and implementation of global standards and solutions to improve the efficiency and visibility of supply and demand chains globally and across sectors. The GS1 system of standards is the most widely used supply chain standards system in the world. GS1 standards bring together companies representing all parts of the supply chain: manufacturers, distributors, retailers, hospitals, transporters, customs organizations,

\(^{48}\) source IDC Manufacturing Insights: Supply Chain in the cloud, looking for beyond total cost of ownership, Nov 2011

\(^{49}\) source SupplyChainBrain.com: The Cloud and the Supply Chain, Mar 2012

\(^{50}\) source Gartner: Supply Chain Advisory, A Cloud Overview, Oct 2011

\(^{51}\) source Gartner: Supply Chain Advisory, Mfg in the Cloud Overview, Feb 2012

\(^{52}\) http://www.gs1.org
software developers, local and international regulatory authorities, and more.

**Figure 5. The GS1 standards.**

- **GS1 Identification Keys:** used to name and distinguish any object, thing or location, so interested parties can get information or business messages related to them.
  - GTIN - Global Trade Item Number
  - SSCC - Serial Shipping Container Code
  - GLN - Global Location Number

- **GS1 Data Carriers:** different kinds of media that can hold GS1 ID Keys and attribute data.
  - EAN/UPC Bar Codes
  - EPC/RFID Tags

- **GS1 Communication Standards:** deal with transactional data, which is data that:
  - acknowledges the completion of a business transaction such as one supported by GS1 eCom standards;
  - deals with visibility data, which is data that logs the occurrence of a physical event (item loaded onto ship, item arrived in port, item inspected at customs...) in EPCIS;
enables master data sharing between trading partners in the supply chain using the Global Data Synchronisation Network (GDSN).

GS1 eCom stands for electronic communication using standard business messaging. GS1 eCom is the GS1 term for Electronic Data Interchange (EDI): the interchange of structured data according to agreed message standards, by electronic means. GS1 eCom features are:

- it provides a standardized and predictable structure of the electronic business messages, enabling business partners to communicate business data rapidly, efficiently and accurately, irrespective of their internal hardware or software types;
- business partner do not have to align the format and structure of the messages - they can use the standard, readily-available format instead;
- it is based on globally recognized GS1 Identification Keys (GTIN, GLN, SSCC etc.);
- It is widely used globally.

Our main work is to transform ITChain to a Collaborative Platform at business level as follows:

Figure 6. GS1 eCom fields.
7.2.4 Towards a SOA approach: the implementation of Supply Chain Management Services

The major challenges the logistics industry must address through improved IT systems are:

- Multi-enterprise collaboration: smarter commerce through end-to-end interoperability of systems and improve distributed planning through the definition and enactment of logistic business processes:
  - Supplier Connection: Cloud based, social business media solution to connect small businesses to large corporations;
  - Global Client Support: End to end view of global clients records across the enterprise from client relationship management to order fulfillment to accounts receivable through finance.

- Multi-enterprise visibility: real-time information availability, cloud based data and analytics:
  - Bundling visibility and control of the value chain into a synchronized portal view;
  - Management by exception. Issue alerts, recommendation, when key performance indicators are trending toward tolerance boundaries;
  - Initiating recovery responses either automatically or through decision feedback loops.
• Dynamic formation of supply chains:
  - Business processes as-a-service;
  - Virtual enterprises;
  - Definition and enactment of integrated business processes.

To link how IT infrastructure testing determines an improvement in operational supply chain management, we propose to establish a trace between SCOR general logistics indicators (Level 1-2-3) with technology (Level 4-5) as shown in the following image:

![SCOR hierarchy](image)

In particular, we establish links between indicators of improved IT infrastructure to SCOR KPI’s Measure Performance:

- Perfect Order Fulfillment;
- Order Fulfillment Cycle Time;
- Supply Chain Flexibility;
- Supply Chain Adaptability;
- Supply Chain Management Cost;

As a summary, we have to monitor the improvement of these Supply Chain Competitive Attributes:

- Reliability: consistently getting the orders right, product meeting quality requirements;
• Responsiveness: the consistent speed of providing products/services to customers;
• Agility: the ability to respond to changes in the market (external influences);
• Cost: associated with managing and operating the supply chain;
• Assets: the effectiveness in managing the supply chain’s assets in support of fulfillment.
7.3 Example: The Interbank Exchange Network

7.3.1 Introduction

The Interbank Exchange Network (IEN) use case is drawn from the section ‘Annex D – Examples’ of the OMG UML Testing Profile (UTP) specification [78] where it is described in detail. We give a concise service oriented perspective of the use case and sketch some samples of the essential service model, the SUT model and the TC model.

The picture Figure 9 (from the UML Testing Profile V1.2 specifications) gives an overview of the system. The Automated Teller Machine (ATM) interconnects to the European Union Bank (EU Bank), through the SWIFT network, which plays the role of a gateway. The example is motivated using an interbank exchange scenario in which a customer with a European Union bank account wishes to deposit money into that account from an ATM in the United States.

Figure 9. Overview of the Interbank Exchange Network (IEN).

The mappings and transformations between the models and the systems are sketched in the diagram Figure 10. From the point of view of the testing activity, and therefore of the MIDAS facility, the system model is a descriptive architectural model of the services architecture under test and the test configuration model is a prescriptive architectural model of the test execution system.
The instantiated services architecture under test is composed of a number of SUT nodes that put into operation the services specified by a service PSM, for instance on the SOAP platform where it is formulated as a WSDL/XDS document. The Service PSM is a manifestation of the service PIM and its elements are deployed on the SUT.

The essential service PIM elements are used to classify the elements of the SUT model. This model is an abstraction of the SUT, i.e. a description of its topology and a documentation of the node/port locations (the locations are expressed as properties of the ports, and are not made explicit in the figures below).
The test configuration platform independent model (TC PIM) is a partial model of the test execution system. It conveys architectural information about the SUT and its testing targets, i.e. it identifies, describes and localizes the SUT nodes/ports that shall be the target of the stimulation, response and observation of the test execution system. The essential service PIM elements classify the TC PIM elements. The TC PIM is obtained by user-driven *model-to-model* (M2M) transformation from the SUT model.

The TC PIM is transformed into a platform specific, implementable test configuration model by an automatic model-to-text (M2T) transformation. In this example, the *specific* platform is a TTCN-3 execution environment, and the TC PSM is a TTCN-3 code artifact, which is a *manifestation* of the TC PIM. Its deployment on the MIDAS TTCN-3 execution environment puts into operation a configuration of the test execution system that takes into account the testing targets of the SUT.

All these models, including the implementable ones, are *architectural models*. As such, they are used by the MIDAS test schemes’ generation and running behaviors, but are independent from any testing approach, method, practice implemented by these behaviors.

The section 7.3.2 (Essential service platform independent model) presents the essential service PIM and its elements such as the *Participant* and the *Service Contract skeletons* through some samples from the IEN use case. The Essential Service PIM is a standard UML model extended by the SoaML profile [77][63]. In order to keep this presentation concise, the associated Service PSM is not detailed.

The section 7.3.3 (System under test model) sketches the SUT platform independent model and its elements, such as the *Node* instances and the *Communication Paths* through these instances, through samples from the IEN use case.

The section 7.3.4 (Test configuration model) introduces the TC platform independent model, the MIDAS Profile and Meta-model elements and some MIDAS test configuration patterns. These issues are illustrated through some samples from the IEN use case.

In the conclusion (0) we summarize and exemplify the model mapping and transformation cycle, from the system under test to the test execution system, via the descriptive system model and the prescriptive test configuration model, through the IEN example.
7.3.2 Essential service platform independent model

The essential service platform independent model of a services architecture under test is a UML/SoaML model [77][63]. It includes a SoaML Service Contract skeleton for each involved service, and a SoaML Participant skeleton for each involved participant system. The SoaML Services Architecture element is used to model the construction of a composite Participant, starting from the top level system.

The shapes of the Service Contract and Participant skeletons will be clarified below. Service Contracts and Participants will be used to classify SUT PIM elements (Node instances) and TC PIM elements (Component instances).

7.3.2.1 Definitions of system, system construction and subsystem

For the purpose of the MIDAS model-based testing approach, a system is constructed with same-meta-type elements (nodes that implement Participant roles) that are linked by same-meta-type links (communication paths that implement Service Channels). Note that, if a communication path is specified between two nodes, this means not only that the involved participant systems are potentially able to implement the appropriate service roles, but also that these roles are actually bound each other by an active service relationship, i.e. by a service relationship that is active in the end-to-end service exchange to be tested, whose interactions are conveyed precisely by the specified communication path. We consider that between two nodes we have as many communication paths as many active service relationships.

In the diagram Figure 11, a “generic” system construction is illustrated by: (i) its composition, i.e. the set of nodes that are “part of” the system inside the system composition boundary, (ii) its environment, i.e. the set of nodes that are not “part of” the system but are connected with at least one composition node and (iii) its structure, the collection of communication paths between all the nodes of the union of the composition set and the environment set. The other nodes, outside the system environment boundary, are not in the system construction.
A more precise formal definition of system and system construction is presented below [18]:

Let $\Pi$ a class of nodes.
Let $\bowtie$ a binary relationship between nodes; for $x, y \in \Pi$, $x \bowtie y$ represents a communication path between $x$ and $y$; $\bowtie$ is symmetric, non-reflexive and non-transitive.
Let $\Sigma$ be a class of systems.
Let $<$ a binary relationship between a node and a system; for $x \in \Pi$ and $\sigma \in \Sigma$, $x < \sigma$ means that $x$ 'is part' of $\sigma$.
The construction of $\sigma$ is a triple $< C(\sigma), E(\sigma), S(\sigma) >$ where:
the composition $C$ of $\sigma$ is defined as $C(\sigma) = \{ x \in \Pi \mid x < \sigma \}$;
the environment $E$ of $\sigma$ is defined as $E(\sigma) = \{ y \in \Pi \mid y \notin C(\sigma) \ \exists x: x \in C(\sigma) \ x \bowtie y \}$;
the structure $S$ of $\sigma$ is defined as $S(\sigma) = \{ < x, y > \mid (x \bowtie y) \ (x, y \in C(\sigma) \ (x \in C(\sigma) \ (y \in E(\sigma)))) \}$. 

The statements above allow a more precise definition of the often ill-defined notion of subsystem:
7.3.2.2 Top level system specification

The diagram Figure 12 presents the specification of the construction of the top level system (InterbankExchangeNetwork) through a Services Architecture. The meanings and uses of Services Architecture and of its parts (classified by Service Contracts and Participants) are defined by the SoaML Profile and Metamodel [63].
The Services Architecture InterbankExchangeNetwork includes four roles that are bound with four Participants – three composition roles and one environment role:

- **:HWControl** - a human or artificial agent operating as an ATM (Automatic Teller Machine) user – this is an environment role bound to Participant HWControl; in UML terms, this is a shared aggregation [63].
- **:ATM** - the Automatic Teller Machine system. This is a composition role bound to Participant ATM.
- **:Bank** - the bank system. This is a composition role bound to Participant Bank.
- **:SwiftNetwork** - the Swift gateway. This is a composition role bound to Participant SwiftNetwork.

In this example, the roles are unnamed, i.e. are indirectly named with their Participant classifiers’ names. The multiplicity of each role (the default multiplicity is 1) specifies the number of Participant parts that can be included in an instance (Collaboration Use) of the Services Architecture InterbankExchangeNetwork.

The relationships among the roles are typed by Service Contracts. This is represented by Dependency relationships between roles and Service Contract parts:

- **:ATMHandling** - regulating the service exchange and the interactions between :HWControl and the :ATM. It is a many-to-many service.
- **:BankService** - defining the service that the :Bank provides to the :ATM. It is a many-to-one service (the multiplicity 1 is default in the diagram).
- **:SwiftExchange** – regulating the service exchange interaction between the :Bank and the :SwiftNetwork. This is a many-to-one service.

In this example, also the Service Contract parts are unnamed, i.e. indirectly named by the Service Contracts that classify them.

The business process through which a customer deposits money into her/his account handled by a EU bank from an US ATM, passing through the Swift network, is choreographed by the UML Activity
DepositFrom_ATM_ThroughSwift. We will not detail this Activity that is not part of the essential service PIM.

7.3.2.3 Service contract skeletons

The “minimal” Service Contracts (skeletons) include only the information needed to build the SUT and TC models. Obviously, the result of a model-driven service engineering cycle could be a richer Service Contract, including not only the skeleton model elements, but also detailed interfaces, protocols, choreographies, operation semantics (Capabilities) and so on.

We present two Service Contract skeleton samples: of a unidirectional service (Figure 13) and of a bidirectional service (Figure 14). A unidirectional service specifies only one Interface that is required by one party and provided by the other party. A bidirectional service specifies two Interfaces that are alternately required and provided by the two parties. The Interface operations can be synchronous request/response or one-way. The meanings and uses of the Service Contract, Consumer, Provider, Service Interface, Capability elements are defined by the SoaAML Profile and Metamodel [63] Interface elements are standard UML interfaces [77].
The elements of the *Service Contract skeleton BankService* (that are in bold lines Figure 13) are:

- The *Service Contract BankService*;
- the role *bankuser* that is bound with the *Consumer BankUser*;
- the role *bankdesk* that bound with the *Provider BankDesk*;
- the *Provider BankDesk* that realizes the Interface *IBank*;
- the *Consumer BankUser* that uses the Interface *IBank*.
- the *Interface IBank* that can be an empty element - it is pointed by (one of) its *manifestations*, the PSM interface IBank.wsdl on the SOAP platform.

Other elements that are not in the *ServiceContract BankService* skeleton could be:

- the *Protocol State Machine BankDesk_Protocol* that defines the interaction protocol with the *Provider role bankdesk*;
- the *Capability BankDesk_Capability exposed* by the *Provider BankDesk* that encapsulates the semantics of the service operations.

The diagram of the bidirectional *Service Contract skeleton ATMHandling* is illustrated Figure 14. Instead of the *Provider* and the *Consumer* that classify the *roles* of a unidirectional *Service Contract*, the *roles atmuser* and *atmcashier* are classified respectively by the *Service Interfaces ATM_User* and *ATM_Cashier*. 
7.3.2.4 Participants

Participants are classes of abstract systems that are used as classifiers of the roles of a Services Architecture. Each Participant is equipped with a port for each Service Contract role that it fulfills, these ports being typed by the service interface elements such as Providers, Consumers (for unidirectional services) and Service Interfaces (for bidirectional services) that are used as classifiers of the roles defined in the Service Contracts that the Participant fulfills. These ports expose the appropriate provided and required interfaces specified by their service interface types.
The collection of Participants involved in the Services Architecture that realizes the construction of the top level InterbankExchangeNetwork composite Participant is presented Figure 15.

**Figure 15. IEN – InterbankExchangeNetwork construction Participants.**

Bank is in turn a composite Participant whose construction is realized by the Services Architecture Bank (Figure 16) that includes two composition roles classified by:

- **Participant BankCounter** – that operates as a “front office”,
- **Participant AccountHandler** – the “back office” that handles the bank accounts\(^{53}\).

\(^{53}\) This internal architecture is introduced here for explanation purposes and does not correspond exactly with that described in the Annex D of the OMG UML Testing Profile (UTP) specification [78].
The other roles (:ATM, :SwiftNetwork) are _environment roles_, i.e. placeholders for systems that are not in the _composition_ of Bank but in its _environment_.

![Diagram](image)

**Figure 16. IEN – Participant Bank construction Services Architecture.**

Participants BankCounter and AccountHandler are presented Figure 17.

![Diagram](image)

**Figure 17. IEN – Participants BankCounter and AccountHandler that fulfil the composition roles of Bank.**
The composite aggregation structure of the Participant Bank is illustrated Figure 18. The composite parts :BankCounter and :AccountHandler are bound with the corresponding roles defined in the Services Architecture Bank (Figure 16). The delegation connectors link the ports of the composite class to the ports of their component parts. So, the port Bank/bankdesk delegates to the port BankCounter/bankdesk and the port Bank/bankentry delegates to (provided interface) and is delegated from (required interface) the port BankCounter/bankentry. The ports linked by the delegation connectors must be classified by the same service interface (Provider, Consumer and Service Interface) elements. The Service Channel Connector links the component Participants’ ports bound with the roles defined in the Service Contract AccountHandling.

Figure 18. IEN – Composite Participant Bank.

All the IEN Participants, with their composite aggregation and active service relationships are presented Figure 19.
Figure 19. IEN - Participant composite aggregation hierarchy and active service relationships.

Figure 20. IEN - Relay Participants.
The essential service model includes other Participants that model service relays. For each Service Contract, a relay Participant is derived that is able to enact all and only the Service Contract roles. Conventionally, we name each relay Participant by the Service Contract from which is derived. The relay Participants model elements can be mechanically generated in a seamless way from the related Service Contracts model elements. Relay Participants will be used to classify TC PIM Interceptor instances (see below). The relay Participants for the IEN use case are presented Figure 20.

7.3.3 System under test model

The system under test model (SUT model) describes the nodes/ports of the deployed SUT that are potentially accessible by the test execution system. It models only the composition of the SUT, not its environment. This model is completely independent from the test schemes that can be applied to the SUT. It describes only its architectural configuration, i.e. its topology, and documents the locations of the deployed and accessible nodes/ports. It does not include any information about test approaches, methods and practices implemented by specific test schemes. It is formulated as a UML Deployment and each deployed and accessible SUT node is represented as a UML Node instance.

A SUT model is always grounded upon an essential service model. The service model Participants are used as classifiers of the SUT model Node instances. The classifiers of the Node instance usbank are presented Figure 21.

The SUT nodes/ports that are described in the SUT model are accessible by the test execution system that can interact directly with them (see below). Conversely, deployed SUT nodes that are not described in the SUT model are not accessible: they cannot interact directly with the test execution system.

Each Node instance defines its accessible ports that are included in the collection of ports that are defined by its Participant classifier. The port locations are documented as port Property values (that are not represented in the figure below).
Accessible connections between SUT nodes are represented by Communication Paths between the Node instances’ ports. Conversely, implemented connections between SUT nodes that are not represented as Communication Paths between the SUT model Node instances are not accessible by the test execution environment.

The SUT model of the IEN use case - in which a customer with a European Union bank account deposits money into that account from an Automated Teller Machine of a United States bank - is illustrated with the deployment diagram Figure 22. It shows five Node instances and four Communication Paths. The Node instances are:

- **usatm**:ATM models the US ATM where the user deposits money – it exhibits two accessible ports (**atmcashier** and **bankuser**);
- **usbank**:Bank models the US bank system – it exhibits two accessible ports (**bankdesk** and **bankentry**);
- **swift**:SwiftNetwork models the Swift exchange system – it exhibits the accessible port **swift**;
- **eubankco**:BankCounter models the EU bank system counter (gateway) component – it exhibits two accessible ports (**bankentry** and **fronoffice**);
• **euaccnth:AccountHandler** models the EU bank system account handler component – it exhibits the accessible *port backoffice*.

![Diagram of IEN - SUT Model](image)

**Figure 22. IEN - SUT model.**

The *ports* are connected by four *Communication Paths* (*cp01, cp02, cp03, cp04*), as indicated in the diagram Figure 22. The *port atmcashier* of *usatm* is not bound to any *Communication Path*. This means that if the test execution system wants to trigger or capture some behavior at this *port* must stimulate or observe it directly.

Note that the *Node* instance *usatm* whose classifier is the composite *Participant Bank* is modeled as a *black-box*, i.e. the test environment cannot access its internal interactions. Conversely, the composite architecture of *Bank* is flattened for the EU bank, and the *Node* instances *eubankco:BankCounter* and *euaccnth:AccountHandler* are explicitly represented with the *Communication Path* between them.

The SUT model *must* be a *flat model*: the nested composites *must* always be flattened to the chosen/allowed local aggregation level. It represents a deployment of the instantiated *Services Architecture* that is presented in the diagram Figure 23. This is a *system construction* model as defined paragraph *Errore. L'origine riferimento non è stata trovata.*
The end-to-end service exchange interactions that are choreographed by **DepositFrom_ATM_ThroughSwift** are accessible on the SUT whose model is presented in the diagram Figure 22. If appropriately configured, the test execution environment is able to send a stimulus through the interface **IATM** of the port **atmcashier** and to access the SUT response at the other accessible **ports** and **Communication Paths**.

### 7.3.4 Test configuration model

The Test Configuration model (TC model) describes the configuration of the architectural components of a UTP **Test Context** [78]. It represents: (i) the SUT nodes and ports that are the targets of stimuli, responses and observations in the course of the test run, (ii) the “sensors/actuators” that are virtually placed on the SUT nodes/connections in order to exercise the aforementioned stimuli, responses and observations and (iii) the connections between SUT and the test execution environment that have to be established. The TC model **must** be associated to a SUT model and **must** be consistent with its topology, but is not aware of the test approaches, methods and practices that are implemented by the test schemes invoked on the SUT: a TC model may be reused by several test schemes.
The TC model is formulated as UML Component model extended by the MIDAS UML Profile and Meta-model, that is an extension of the UTP Profile and Meta-model [78]. The elements of the TC model are stereotyped Component instances.

7.3.4.1 TC UML profile

MIDAS introduces three UML Component stereotypes (Figure 24) as a specialization of the UTP profile and meta-model [78]. The MIDAS Proxy, Emulator and Interceptor specialize the UTP TestComponent as an extension of Component.54

Figure 24. Test configuration profile.

A Proxy Component models a test execution system component that operates as a proxy of an accessible SUT node - described by a SUT model Node instance - that is the potential target of stimuli, responses and observations of the test execution environment.

An Emulator Component models a test execution system component that is able to emulate either a SUT node (represented by a UML Node instance in the SUT model), which is a composition element, or a “virtual” environment node. For instance, an environment node for the SUT deployment represented Figure 22 could be an implementation of the Participant HWControl, i.e. a system that implements the Service Interface ATM_User on a port such as atmuser.

---

54 This is in fact a restricted specialization of the UTP profile, in which TestComponent is an extension of the more general Class classifier.
The test execution system component modeled by an *Emulator* instance shall be able to interact with the SUT nodes through the test execution system component modeled by the corresponding *Proxy* instance, and the allowed interactions between the test execution environment and the SUTs nodes must be formally compliant with the service interfaces specified by the *Participants* that classifies respectively the *Emulator* instance and the *Proxy* instance.

An *Interceptor Component* models a test execution system component that is able to act as if it were virtually placed on an accessible communication path between SUT nodes that is represented by a *Communication Path* between ports in the SUT model.

The test execution system component modeled by an *Interceptor* shall be able to interact with the SUT nodes at the connection end-points, and the resulting interactions shall be compliant with the service interfaces specified by the *Participants* that classifies the end-point SUTs.

The TC model elements instantiate the essential service model *Participants*. *Proxy* instances must be classified only by *Participants* that classifies SUT model *Node* instances. The classifier of an *Emulator* instance must be one of the *Participants* that classify composition and environment roles in the corresponding instantiated *Services Architecture*. *Interceptor* instances must be classified by the relay *Participants* derived by the Service Contracts involved in the aforementioned *Services Architecture*.

*Proxy* instances model test execution system components that must act as mere proxies of the SUT nodes and shall not implement any particular testing operational semantics. The *Emulator* and *Interceptor* instances are *per se mere skeletons* without testing operational semantics. They are used to configure the capability of the test execution system to interact with the SUT in compliance with the service interface specifications documented in the service model. Their particular operational semantics in a *test scenario*, i.e. the interaction control flow, the interaction contents and the actions at test run time, is provided by the invoked test scheme and represented by a UTP *Test Case*.

### 7.3.4.2 Test configuration patterns

We introduce three *test configuration pattern* families:

- *unit test configuration* patterns,
- *unit test with stubs configuration* patterns,
• **integration test configuration** patterns.

Figure 25 sketches an example of the application of the *unit test configuration pattern* for bidirectional services to the SUT whose model is illustrated by the diagram Figure 22.

![Figure 25. IEN - TC model - unit test.](image)

The *Emulator* instance and the *Proxy* instance that are specified in the TC model Figure 25 are classified by the service model *Participants* as shown Figure 26 and Figure 27.

![Figure 26. IEN - Emulator hwcntrl classifiers.](image)
Figure 27. IEN - Proxy euatm classifiers.

The TC model Figure 25, produced by the application of the unit test pattern, specifies the stimulation and the observation of the behavior of the SUT usatm through the exchange regulated by the bidirectional service ATMHandling through the accessible port atmcashier classified by the Provider ATM_Cashier.

The diagram Figure 28 documents the result of the application of a unit test with stubs configuration pattern to the IEN SUT (Figure 22). The behavior of the node represented by the Proxy usatm:ATM is stimulated and observed at the ports atmcashier by the Emulator instance hwnctrl:HWControl and bankuser by the Emulator instance usbank:Bank.

This pattern is well suited for unit testing within service compositions. In order to provide the service ATMHandling to hwnctrl - for instance to effect a synchronous withdrawal from a US account through a US ATM - usatm needs to consume the service BankService provided by usbank. Hence, an interaction at the port atmcashier can cause an interaction at the port bankuser. If the service model of the Participant ATM includes the definition of transfer functions that specifies the correspondence between IBank operations’ input variables with the corresponding (in the service composition) IATM operations’ input
variables\textsuperscript{55}, advanced testing methods allow an appropriated \textit{Emulator usbank:Bank} to get \textit{static or dynamic oracles} for IBank interactions.

![Diagram of IEN – TC model – unit test with stub](image)

\textbf{Figure 28. IEN – TC model – unit test with stub.}

The diagram Figure 29 \textit{Errore. L’origine riferimento non è stata trovata.} documents a TC model produced by the application of an \textit{integration test configuration pattern} to the IEN SUT (Figure 22).

![Diagram of IEN – TC model – integration test](image)

\textbf{Figure 29. IEN – TC model – integration test.}

\textsuperscript{55} For instance, the transfer function can be formulated in OCL.
The diagram Figure 30 shows the classifiers of the *Interceptor* instance *cp01* that is employed in this model.

The *Interceptor* instance *cp01* allows specifying the observation of and the actions on the interactions between the port *usatm/bankuser* and the port *usbank/bankdesk*, i.e. the integration test between *usatm* and *usbank*.

### 7.3.4.3 The Interbank Exchange Network TC model

The combination of the patterns presented in the preceding section should allow the definition of almost any test configuration. The end-to-end service exchange that realizes business process of the deposit of money in an EU bank account through a US ATM, can be stimulated and observed by the appropriate combination of the test configuration patterns on the SUT.

The TC model is the result of a M2M mechanical transformation from the SUT model that is driven by the tester specification. The tester specifies: (i) the SUT *subsystem*, i.e. the collection of the SUT *Nodes* (the *Proxy* instances) that are the targets of stimuli, responses and observations (and, implicitly, the connections between SUT and the test execution environment); (ii) the *Emulator* instances that either replace SUT *Nodes* or implement “virtual” environment nodes; and (iii) the *Interceptor* instances that are virtually placed on the SUT *Connection Paths*. 
A TC model for partial integration test on the SUT whose model diagram is presented Figure 22, is illustrated Figure 31. This TC model specifies a test configuration in which only the US system is tested (the Swift network is replaced by an Emulator instance).

![Diagram](image.png)

**Figure 31. IEN – Partial test integration configuration model.**

A TC model for end-to-end service exchange on the SUT whose model diagram is presented Figure 22, is illustrated Figure 32. This TC model includes five Proxy instances, one Emulator instance and four Interceptor instances. It specifies a complete integration test execution system in which a ‘virtual’ environment node is implemented by an emulator and all the other SUT nodes are observed/stimulated by interceptors through the proxies.
Each test scheme that conforms to UTP should be able to build the Test Context by M2M transformation of the TC model, i.e. by filling the TC model with a specific Test Case, and subsequently to generate the test execution environment specific code artifact (in this example the TTCN-3 code) by M2T transformation.

**7.3.5 Conclusion**

An IEN example of model mapping and transformation is sketched in the diagram Figure 33. The implemented TC model is that illustrated by the diagram Figure 31.
Note that: (i) the notations that are used in Figure 33 to sketch the SUT and the test execution system are the same that are employed for the formulations of respectively the SUT model and the TC model; (ii) the communication path cp01 is actually cut by the test configuration, because the interactions between port usatm/bankuser and port usbank/bankdesk nodes are intercepted by the test execution system, in compliance with the TC model illustrated in Figure 31; (iii) the ports of proxies usatm and usbank are bound to the corresponding ports of the namesake SUT nodes.

Figure 33. IEN – Model mapping and transformation cycle.
With the IEN use case illustrated Figure 33 we exemplify the model mapping and transformation cycle that conducts to the deployment of the architectural code of a test execution system in the MIDAS facility.

In the example the system model includes an essential service PIM and a service PSM (a collection of WSDL artifacts). These service PIM/PSM are the results of: (i) either a contract-based, model-driven engineering cycle, (ii) or a reverse modeling activity, (iii) or both. On the basis of these models, the MIDAS user (for instance, the testee) is able to author the SUT model and upload it on the MIDAS facility in a seamless way.

From the descriptive system model (PIM, PSM and SUT models) it is possible to obtain, by M2M transformation driven by the tester specification, the prescriptive TC PIM with a very limited effort of the user.

The M2T transformation from the TC PIM to the executable TC PSM on the TTCN-3 execution environment, with automated generation of the test execution system skeleton code (its “architectural” part), is performed directly by the MIDAS facility. Each test scheme generation behavior should be able to: (i) transform the TC model in a full Test Context model by combining it with the Test Case that it realizes; (ii) generate from the Test Context the platform specific test execution system (architecture and operational semantics), on one or more test execution platforms such as the TTCN-3 test execution environment, by applying its own methods and by using the MIDAS test execution system configuration API.

In terms of reuse, starting from the essential service PIM and the service PSM, the user can author several SUT models, corresponding to different SUT deployments. Starting from a SUT model and the associated service PIM/PSM the user can specify and generate several TC models by user-driven M2M transformation. Moreover, each TC model, that is an architectural model independent by any test method and approach, can be reused by each MIDAS compliant test scheme that is able to build a Test Context by M2M transformation. Last but not least, the generation of the test execution system by M2T transformation of the Test Context is proposed by MIDAS for the TTCN-3 platform, but is not a priori limited to that platform.

When the service model that is issued from the design phase lacks some features of the essential service model it must be completed by the user.
REFERENCES


Deliverable D2.1


