



MADES Communication Standard

2014-06-20

VERSION 1.1

| | | Table of Contents | |
|----|--------|--|----|
| 2 | | | |
| 3 | 1 | Introduction | 9 |
| 4 | 1.1 | What is MADES about? | 9 |
| 5 | 1.2 | MADES Governance | 9 |
| 6 | 2 | High level concepts | 10 |
| 7 | 2.1 | What is MADES intended for? | 10 |
| 8 | 2.2 | What MADES is not? | 11 |
| 9 | 2.3 | Used Acronyms | 14 |
| 10 | 2.4 | General overview | 15 |
| 11 | 2.5 | Message delivery and transparency | 15 |
| 12 | 2.5.1 | Message delivery | 15 |
| 13 | 2.5.2 | Transparency | 16 |
| 14 | 2.6 | Security and reliability | 16 |
| 15 | 2.7 | Main components | 17 |
| 16 | 2.8 | Distributed architecture | 18 |
| 17 | 2.9 | Components' exposed interfaces | 18 |
| 18 | 2.10 | Security features | 19 |
| 19 | 2.10.1 | Overview | 19 |
| 20 | 2.10.2 | Transport-layer security | 20 |
| 21 | 2.10.3 | Message-level security | 20 |
| 22 | 2.10.4 | Non repudiation | 22 |
| 23 | 3 | Components' functions | 23 |
| 24 | 3.1 | Routing messages | 23 |
| 25 | 3.2 | Component and message unique identification (ID) | 24 |
| 26 | 3.3 | Business-type of a business-message | 24 |
| 27 | 3.4 | Delivery-status of a business-message | 25 |
| 28 | 3.5 | Communication between components | 26 |
| 29 | 3.5.1 | Principle | 26 |
| 30 | 3.5.2 | Establishing a secured communication channel between two | |
| 31 | | components | 26 |
| 32 | 3.5.3 | Token authentication of the client component | 27 |
| 33 | 3.5.4 | Request authorisation | 27 |
| 34 | 3.5.5 | Request/Reply validation | 28 |
| 35 | 3.6 | Storing messages in components | 28 |
| 36 | 3.7 | Lifecycle of a message state within a component | 28 |
| 37 | 3.8 | Transferring a message between two components (Handshake) | 30 |
| 38 | 3.9 | Accepting a message | 31 |
| 39 | 3.10 | Event management | 32 |
| 40 | 3.10.1 | Acknowledgements | 32 |
| 41 | 3.10.2 | Notifying events | 33 |
| 42 | 3.10.3 | Lifecycle of an acknowledgement | 35 |
| 43 | 3.10.4 | Processing a transferred acknowledgement | 35 |
| 44 | 3.11 | Message expiration | 36 |
| 45 | 3.11.1 | Principle | 36 |
| 46 | 3.11.2 | Setting the expiration time of a message: | 36 |
| 47 | 3.11.3 | Looking for the expired messages: | 36 |
| 48 | 3.12 | Checking the connectivity between two endpoints (Tracing-messages) | 36 |

| | | | |
|----|--------|---|----|
| 49 | 3.13 | Ordering the messages (Priority) | 37 |
| 50 | 3.14 | Endpoint | 37 |
| 51 | 3.14.1 | Endpoint functions | 37 |
| 52 | 3.14.2 | Compression | 38 |
| 53 | 3.14.3 | Signing | 39 |
| 54 | 3.14.4 | Encryption | 40 |
| 55 | 3.15 | Node | 41 |
| 56 | 3.15.1 | Node functions | 41 |
| 57 | 3.15.2 | Synchronizing directory with other nodes | 42 |
| 58 | 3.15.3 | Updating the synchronization nodes' list | 43 |
| 59 | 3.16 | Certificates and directory management | 43 |
| 60 | 3.16.1 | Definitions and principles | 43 |
| 61 | 3.16.2 | Certificates: Format and unique ID | 45 |
| 62 | 3.16.3 | Used certificates and issuers (CAs) | 45 |
| 63 | 3.16.4 | Directory services | 46 |
| 64 | 3.16.5 | Caching directory data | 47 |
| 65 | 3.16.6 | Trusting the certificates of others components | 47 |
| 66 | 3.16.7 | Renewing the expired certificates | 47 |
| 67 | 3.16.8 | Revoking a certificate | 48 |
| 68 | 4 | Managing the version of the MADES specification | 49 |
| 69 | 4.1 | Issues and principles | 49 |
| 70 | 4.1.1 | General | 49 |
| 71 | 4.1.2 | Rolling out a new version (Mversion and N-compliance) | 49 |
| 72 | 4.1.3 | Service compatibility | 50 |
| 73 | 4.1.4 | Message compatibility | 50 |
| 74 | 4.1.5 | Interface with BAs | 51 |
| 75 | 4.2 | Using the correct version for services and messages | 51 |
| 76 | 4.2.1 | Node synchronization and authentication | 51 |
| 77 | 4.2.2 | Directory services and Network acceptance | 52 |
| 78 | 4.2.3 | Messaging services | 53 |
| 79 | 4.2.4 | Which version to use to send a message? | 54 |
| 80 | 5 | Interfaces and services | 55 |
| 81 | 5.1 | Introduction | 55 |
| 82 | 5.1.1 | General | 55 |
| 83 | 5.1.2 | Error Codes | 55 |
| 84 | 5.1.3 | Types for Time | 56 |
| 85 | 5.2 | Endpoint interface | 56 |
| 86 | 5.2.1 | Overview | 56 |
| 87 | 5.2.2 | Services | 56 |
| 88 | 5.2.3 | File System Shared Folders (FSSF) | 60 |
| 89 | 5.3 | Node interface | 63 |
| 90 | 5.3.1 | Overview | 63 |
| 91 | 5.3.2 | Authentication service | 63 |
| 92 | 5.3.3 | Messaging Services | 64 |
| 93 | 5.3.4 | Directory services | 68 |
| 94 | 5.3.5 | Node Synchronization interface | 71 |
| 95 | 5.4 | Format of the node-list file | 72 |

| | | | |
|-----|-------|---|-----|
| 96 | 5.5 | Typed Elements used by the interfaces | 72 |
| 97 | 5.6 | Description of the services | 80 |
| 98 | 5.6.1 | About WSDL and SOAP | 80 |
| 99 | 5.6.2 | Endpoint interface | 80 |
| 100 | 5.6.3 | Node interface | 87 |
| 101 | 5.6.4 | XML signature example | 101 |
| 102 | | | |
| 103 | | List of figures | |
| 104 | | Figure 1 – MADES overall view | 10 |
| 105 | | Figure 2 – MADES scope | 11 |
| 106 | | Figure 3 – MADES key features | 15 |
| 107 | | Figure 4 – MADES message delivery overview | 16 |
| 108 | | Figure 5 – MADES security and reliability | 16 |
| 109 | | Figure 6 – MADES components | 17 |
| 110 | | Figure 7 – MADES network distributed architecture | 18 |
| 111 | | Figure 8 – MADES interfaces and services | 19 |
| 112 | | Figure 9 – MADES transport security overview | 20 |
| 113 | | Figure 10 – MADES secure communication initiation | 20 |
| 114 | | Figure 11 – Message signature | 21 |
| 115 | | Figure 12 – Message encryption and decryption | 21 |
| 116 | | Figure 13 – Non repudiation..... | 22 |
| 117 | | Figure 14 – Delivery route of a business-message | 23 |
| 118 | | Figure 15 – Reported events during the delivery of a business-message | 25 |
| 119 | | Figure 16 – Lifecycle of the local state of a business-message within a component | 29 |
| 120 | | Figure 17 – Transfer handshake when uploading of a message | 30 |
| 121 | | Figure 18 – Transfer handshake when downloading of a message | 31 |
| 122 | | Figure 19 – Acknowledgements along the route of the business-message | 33 |
| 123 | | Figure 20 – Encryption process..... | 40 |
| 124 | | Figure 21 – A node synchronizes with two other nodes | 42 |
| 125 | | Figure 22 – Certificates and certificate authorities (CAs) for a MADES network | 45 |
| 126 | | Figure 23 – Managing the specification version – node synchronization and | |
| 127 | | authentication | 52 |
| 128 | | Figure 24 – Managing the specification version – Directory services | 52 |
| 129 | | Figure 25 – Managing the specification version – Messaging services | 53 |
| 130 | | Figure 26 – Managing the specification version – Which version to use to send a | |
| 131 | | message? | 54 |
| 132 | | Figure 27 – Node interface – Overview | 63 |
| 133 | | Figure 28 – Node interface – Authentication service | 64 |
| 134 | | Figure 29 – Node interface – Messaging services – UploadMessages service | 65 |
| 135 | | Figure 30 – Node interface – Messaging services – DownloadMessages service | 66 |
| 136 | | Figure 31 – Node interface – Messaging services – ConfirmDownload service..... | 67 |
| 137 | | Figure 32 – Node interface – Directory services – GetCertificate service | 69 |
| 138 | | Figure 33 – Node interface – Directory services – GetComponent service | 70 |

| | | |
|-----|---|----|
| 139 | Figure 34 – WSDL 1.1 definitions..... | 80 |
| 140 | | |
| 141 | List of tables | |
| 142 | Table 1 – Message delivery status | 26 |
| 143 | Table 2 – Business message status | 29 |
| 144 | Table 3 – Accepting a message – Validation checks | 32 |
| 145 | Table 4 – Characteristics of notified events..... | 33 |
| 146 | Table 5 – Event characteristics description | 34 |
| 147 | Table 6 – Acknowledgement state description..... | 35 |
| 148 | Table 7 – Compression – metadata attributes | 39 |
| 149 | Table 8 – Signing – metadata attributes | 39 |
| 150 | Table 9 – Encryption – metadata attributes | 41 |
| 151 | Table 10 – Consequences of a certificate revocation | 49 |
| 152 | Table 11 – Service compatibility – Possible changes | 50 |
| 153 | Table 12 – Which version to use to send a message? | 54 |
| 154 | Table 13 – Managing the specification version – Rejection conditions..... | 54 |
| 155 | Table 14 – Interfaces and services – Generic error | 55 |
| 156 | Table 15 – Interfaces and services – String value for errorCode | 55 |
| 157 | Table 16 – SendMessage – Service request elements | 56 |
| 158 | Table 17 – SendMessage – Service response elements..... | 57 |
| 159 | Table 18 – SendMessage – Additional error elements..... | 57 |
| 160 | Table 19 – ReceiveMessage – Service request elements..... | 57 |
| 161 | Table 20 – ReceiveMessage – Service response elements | 58 |
| 162 | Table 21 – ReceiveMessage – Additional error elements | 58 |
| 163 | Table 22 – CheckMessageStatus – Service request elements..... | 58 |
| 164 | Table 23 – CheckMessageStatus – Service response elements | 58 |
| 165 | Table 24 – CheckMessageStatus – Additional error elements | 58 |
| 166 | Table 25 – ConnectivityTest – Service request elements | 59 |
| 167 | Table 26 – ConnectivityTest – Service response elements..... | 59 |
| 168 | Table 27 – ConnectivityTest – Additional error elements..... | 59 |
| 169 | Table 28 – ConfirmReceiveMessage – Service request elements..... | 59 |
| 170 | Table 29 – ConfirmReceiveMessage – Service response elements | 60 |
| 171 | Table 30 – ConfirmReceiveMessage – Additional error elements | 60 |
| 172 | Table 31 – FSSF – Description and filename format | 61 |
| 173 | Table 32 – FSSF – Filename description..... | 61 |
| 174 | Table 33 – Authentication – Service request elements | 64 |
| 175 | Table 34 – Authentication – Service response elements | 64 |
| 176 | Table 35 – UploadMessages – Service request elements..... | 65 |
| 177 | Table 36 – UploadMessages – Service response elements | 66 |
| 178 | Table 37 – DownloadMessages – Service request elements | 66 |
| 179 | Table 38 – DownloadMessages – Service response elements..... | 67 |
| 180 | Table 39 – ConfirmDownload – Service request elements..... | 67 |

| | | |
|-----|--|----|
| 181 | Table 40 – ConfirmDownload – Service response elements | 67 |
| 182 | Table 41 – SetComponentMversion – Service request elements | 68 |
| 183 | Table 42 – SetComponentMversion – Service response elements..... | 68 |
| 184 | Table 43 – GetCertificate – Service request elements | 69 |
| 185 | Table 44 – GetCertificate - Service response elements..... | 69 |
| 186 | Table 45 – GetCertificate – Additional conditions | 69 |
| 187 | Table 46 – GetComponent – Service request elements | 70 |
| 188 | Table 47 – GetComponent – Service response elements | 70 |
| 189 | Table 48 – GetNodeMversion – Service request elements | 71 |
| 190 | Table 49 – GetNodeMversion – Service response elements..... | 71 |
| 191 | Table 50 – GetAllDirectoryData – Service request elements | 71 |
| 192 | Table 51 – GetAllDirectoryData – Service response elements..... | 71 |
| 193 | Table 52 – Node attributes ordered list | 72 |
| 194 | Table 53 – AuthenticationToken..... | 72 |
| 195 | Table 54 – Certificate..... | 73 |
| 196 | Table 55 – CertificateType – string enumeration | 73 |
| 197 | Table 56 – ComponentCertificate | 73 |
| 198 | Table 57 – ComponentDescription | 73 |
| 199 | Table 58 – ComponentInformation | 73 |
| 200 | Table 59 – ComponentType – string enumeration | 74 |
| 201 | Table 60 – Endpoint..... | 74 |
| 202 | Table 61 – InternalMessage..... | 74 |
| 203 | Table 62 – InternalMessageType – string enumeration | 75 |
| 204 | Table 63 – MessageMetadata | 76 |
| 205 | Table 64 – MessageProcessor | 76 |
| 206 | Table 65 – Map..... | 76 |
| 207 | Table 66 – MapEntry..... | 76 |
| 208 | Table 67 – ValueType (enumeration) | 76 |
| 209 | Table 68 – MessageState (string enumeration) | 77 |
| 210 | Table 69 – MessageStatus..... | 77 |
| 211 | Table 70 – MessageTraceItem | 77 |
| 212 | Table 71 – MessageTraceState (string enumeration) | 78 |
| 213 | Table 72 – NotConfirmedMessageResponse..... | 78 |
| 214 | Table 73 – NotUploadedMessageResponse | 78 |
| 215 | Table 74 – ReceivedMessage | 79 |
| 216 | Table 75 – RoutingInformation | 79 |
| 217 | Table 76 – SentMessage | 79 |
| 218 | | |

219

Copyright notice:

220

Copyright © ENTSO-E. All Rights Reserved.

221

This document and its whole translations may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, except for literal and whole translation into languages other than English and under all circumstances, the copyright notice or references to ENTSO-E may not be removed.

222

223

224

225

226

227

228

229

This document and the information contained herein is provided on an "as is" basis.

230

231

232

233

ENTSO-E DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

234

Maintenance notice:

235

236

THIS DOCUMENT IS MAINTAINED BY THE ENTSO-E WG EDI. COMMENTS OR REMARKS ARE TO BE PROVIDED AT EDI.Library@entsoe.eu

237

Revision History

238

| Version | Release | Date | Paragraph | Comments |
|---------|---------|------------|-----------|---|
| 1 | 0 | 2012-01-18 | | Approved by the Market Committee. |
| 1 | 1 | 2014-06-07 | | Editorial correction and alignment with the Technical Specification. Submitted to Market Committee approval. |

239

240 1 Introduction

241 The MADES initiative specifies a standard for a communication platform which every
242 Transmission System Operator (TSO) in Europe may use to reliably and securely exchange
243 documents. Consequently a European market participant (trader, distribution utilities, etc.)
244 could benefit from a single, common, harmonized and secure platform for message exchange
245 with the different TSOs; thus reducing the cost of building different IT platforms to interface
246 with all the parties involved. This also represents an important step in facilitating parties
247 entering into markets other than their national ones.

248 The “MADES” acronym is short for: **MArket Data Exchange Standard**.

249 The MADES initiative has been adopted by the ENTSO-E Electronic Data Interchange (EDI)
250 Working Group (WG) to facilitate communication between TSOs and European electricity
251 market participants.

252 The document is published on the ENTSO-E website (<https://www.entsoe.eu>).

253 1.1 What is MADES about?

254 MADES is a specification for a decentralized common communication platform based on
255 international IT protocol standards:

- 256 • From a business application (BA) perspective, MADES specifies software interfaces to
257 exchange electronic documents with other BAs. Such interfaces mainly provide means to
258 send and receive documents using a so-called “MADES network”. Every step of the
259 delivery process is acknowledged, and the sender can request about the delivery status of
260 a document. This is done through acknowledgement, which are messages returned back
261 to the sender. This makes MADES networks usable for exchanging documents in business
262 processes requiring reliable delivery.
- 263 • MADES also specifies all services for the business application (BA); the complexities of
264 recipient localisation, recipient connection status, message routing and security are
265 hidden from the connecting BA. MADES services include directory, authentication,
266 encryption, signing, message tracking, message logging and short-term message storage.

267 The purpose of MADES is to create a data exchange standard comprised of standard
268 protocols and utilizing IT best practices to create a mechanism for exchanging data over any
269 TCP/IP communication network, in order to facilitate business to business information
270 exchanges as described in IEC 62325-351 and the IEC 62325-451 series.

271 A MADES network acts as a post-office organization. The transported object is a “message” in
272 which the sender document is securely repackaged in an envelope (i.e. a header) containing
273 all the necessary information for tracking, transportation and delivery.

274 1.2 MADES Governance

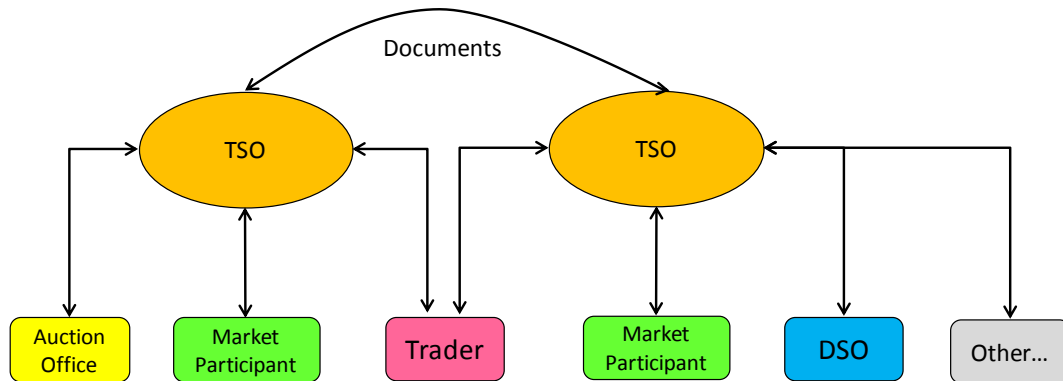
275 ENTSO-E shall continue to maintain the MADES. The aim of the governance is to provide
276 stability in the standard. There will be at most one version released per year, to reduce the
277 burden of change on the user community.

278 MADES version numbering is in full integers. Revisions may only apply to the specification
279 document which could be reissued to fix reported errors or ambiguities. The last issued
280 revision is the only one applicable for a version.

281 MADES specifies rules to allow a smooth rollout process when upgrading the implemented
282 version of an existing operational network — see § 4.

283 **2 High level concepts**

284 **2.1 What is MADES intended for?**



285

286

Figure 1 – MADES overall view

287 MADES' first intention is to provide TSOs with a standardized communication access point to
288 securely exchange documents with others parties involved in the European electricity market
289 as shown in Figure 1. These documents are mainly the ones used in the energy market and
290 described in IEC 62325-351 and the IEC 62325-451 series. Such parties include TSOs,
291 distribution system operators (DSO), balance responsible parties (BRP), capacity traders
292 (CT), market operators (MO), producers, transmission capacity allocators (TCA), etc.

293 MADES is a generic way to exchange information. It is not limited in usage to market data or
294 the electricity industry and can be used more widely for any non-real time data exchange
295 application.

296 The MADES enables each party to implement MADES access points (referred to as
297 endpoints) connected to his information system (IS), where he may securely send and receive
298 documents to and from other parties.

299 MADES is not concerned with specific business functionality, neither creating a new IT
300 standard, nor building communication infrastructure. For the market to operate correctly,
301 parties must exchange electronic documents conforming to predefined business logic and in a
302 cost effective way, namely by using existing IT standards and protocols over existing
303 communication infrastructures.

304 **The purpose of MADES is to create a data exchange standard** comprised of standard
305 protocols and utilizing IT best practices to create a mechanism for exchanging data over any
306 TCP/IP communication network, in order to facilitate business-to-business information
307 exchanges.

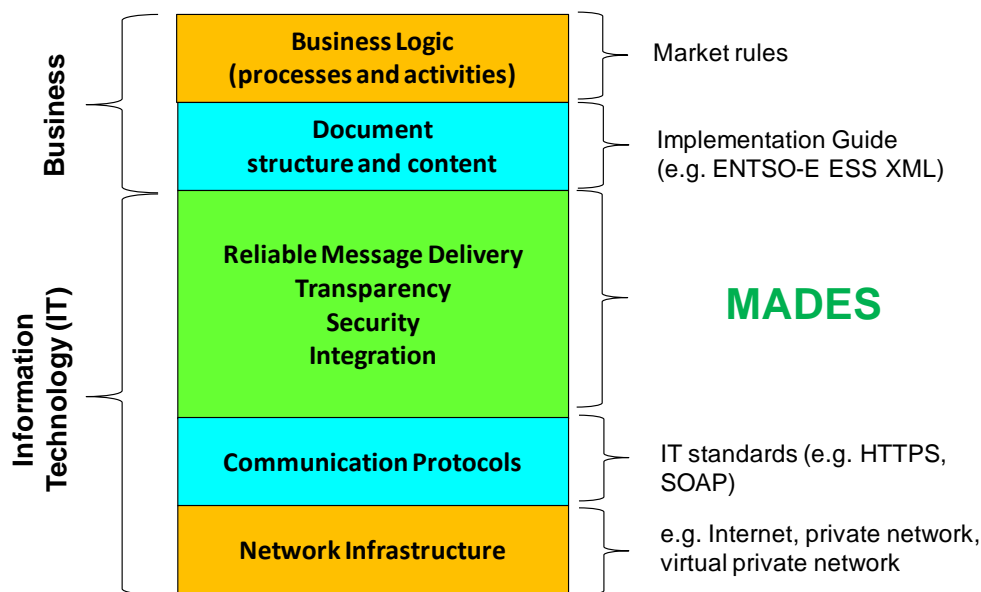
308 New market rules induce new business processes and activities, and generally require new
309 information exchanges between parties. Experience shows that, for the exchanges to operate
310 according to the business goals, the chosen technical solution results from an agreement of
311 involved parties gathering various constraints, including implementation time scale, vendors'
312 offer, already existing communication links, integration capabilities of existing information
313 systems, confidentiality of exchanged information, legal risks, etc.

314 Where business processes require information to be exchanged between multiple systems or
315 multiple parties, solutions developed bilaterally may become extremely complex, with each
316 interface taking time, money and resources to be developed and be maintained. It is also a
317 noticeable consequence that some parties acting in several countries, such as traders, may

318 have to install different communication tools in order to interface with different trading
319 solutions. The future vision is a single interface between all parties in all areas of the
320 electricity market of Europe.

321 MADES is a step forward to a standardized communication solution, especially in the
322 following areas:

- 323 • Future ENTSO-E European projects or TSO domain projects should reduce required
324 resources and time to operate the new market rules (less design, less tests).
- 325 • A MADES access point may be implemented using any software compliant with this
326 publicly accessible specification. Vendors are encouraged to either interface or integrate a
327 MADES compliant client in their Business Applications.
- 328 • European market should be facilitated as multi-countries actors should have a single
329 access to market, or can deploy a unique solution in different sites.



330
331 **Figure 2 – MADES scope**

332 MADES can support any business process whatever the document types being transmitted
333 might be (e.g. XML, binary) and whatever the sequence for the exchanges.

334 MADES is independent of the physical underlying communication Infrastructure, which can be
335 any IP (Internet Protocol) network, such as Internet, a physical private infrastructure, or a
336 multi access-point virtual private network (VPN).

337 MADES relies on and only on non-proprietary IT standards for communication protocols, data
338 integrity, signing and confidentiality (encryption), peer access point authentication, peer party
339 authentication, parties' directory (e.g. HTTPS, SOAP, X.509), as shown in Figure 2.

340 **2.2 What MADES is not?**

341 MADES is not a synchronous messaging system:

342 MADES specifies a framework for asynchronous communication; therefore the architecture
343 contains hubs (namely Nodes) that offer a service to temporary store messages.

344 A synchronous messaging system would require that the sender access point establishes a
345 direct IP connection to the recipient access point for exchanging messages, thus meaning that
346 both have to be online at the same moment.

347 The way MADES achieves a transfer is to send each message to a message queue at the
348 Node where the recipient then retrieves it. As an asynchronous process, there is no direct
349 connection between sender and recipient, i.e. no handshake exists between peer access
350 points.

351 This design has two main benefits:

- 352 • The recipient may be offline for a given duration without losing any information. So, parties
353 not involved in frequent or business critical processes can turn off their access points —
354 actually have it installed on a non-permanently network connected computer.
- 355 • The security level for the architecture is higher than other methods since access points
356 are not required to accept incoming connections, i.e. all connections are initiated by the
357 client and so no exception to firewall rules is required.

358 MADES is not intended for real time messaging:

359 The magnitude for the end-to-end duration of message transportation is not defined and may
360 significantly vary depending on the implementation and the underlying infrastructure due to
361 the following considerations:

- 362 • The transfer process is asynchronous and not event-driven.
- 363 • The exchanged documents can range in size from kilobytes to several megabytes.
- 364 • The security management requires processing resource for signing, verifying
365 signature, encryption, decryption.

366 As a consequence, MADES is not intended and should not be recommended nor considered
367 for processes or projects dealing with real time messaging.

368 MADES is not about designing or delivering software:

369 MADES is concerned with the design of interfaces of access points and hubs (Nodes);
370 however the internal design of those components is of out of scope. Internal design
371 considerations could include:

- 372 • Functional architecture for message management, storage and archiving, security
373 management, directory management.
- 374 • Logical and technical architecture for performance.
- 375 • Redundancy for high availability solution.
- 376 • Software packaging and installation process.
- 377 • Administration tools design and security (e.g. Graphical User Interface).
- 378 • Component supervision agent.
- 379 • Statistics and Key Performance Indicators (KPI) collection.

380 MADES is not about setting up a network:

381 MADES is a specification. A MADES network is a group of parties, each operating
382 communication components (Endpoints, Nodes) which comply with the specification. Setting
383 up a MADES network requires more than software, and a governance team must:

- 384 • choose the underlying network infrastructure,
- 385 • define the network access rules,
- 386 • define the access points identification scheme,
- 387 • define the network joining process (e.g. using a test network first),

- 388 • define which parties can or must host the Nodes,
- 389 • define the network supervision organisation,
- 390 • formalize the Node administrator role and tasks,
- 391 • define and supervise the planning in case of a version upgrade,
- 392 • specify the archiving/logging duration requirements,
- 393 • define the certificate policy which states the roles and duties of the different actors of the
- 394 public key infrastructure, the certificate validity duration, the trusted certificate authorities
- 395 (CAs), the process to revoke and renew the certificates,
- 396 • define the backup and archiving strategies,
- 397 • etc.

398 MADES is not a complete business solution:

399 Refer back to Figure 2 to see what a complete solution would include.

400 Some parties may look for a plug-and-play tool (easy to install, use, administrate, upgrade),
401 where documents can be sent and received using drag-and-drop by selecting the recipient in
402 list, where logs and archives can be easily scanned, scrolled and browsed, where errors send
403 configurable alarms, etc.

404 MADES does not specify any Graphical User Interface; it focuses on interfaces to ensure that
405 access points interoperate. However MADES in no way impedes the creation of a human
406 machine interface layer which may use MADES for message transport.

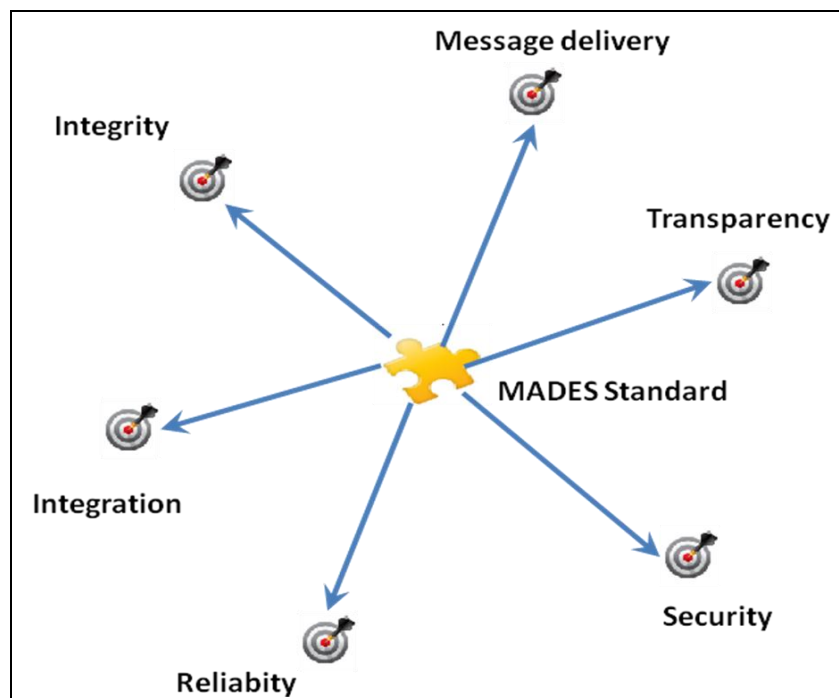
407 **2.3 Used Acronyms**

| | |
|----------------------------|---|
| AES | Advanced Encryption Standard — A symmetric cryptographic algorithm. |
| BA | Business Application |
| DER | Distinguished Encoding Rules — A format for X.509 digital certificates |
| DMZ | DeMilitarized Zone |
| EDI | Electronic Data Interchange |
| FSSF | File System Shared Folder |
| HTTPS | HTTP (HyperText Transfer Protocol) Secured with the TLS protocol to provide encrypted communication and secure identification of a web server. |
| IETF | Internet Engineering Task Force |
| ID | IDentity |
| IP | Internet Protocol |
| IS | Information System |
| IT | Information Technology |
| ITU-T | The standardization sector of the International Telecommunication Union (ITU) |
| PKI | Public Key Infrastructure |
| RFC | Request For Comments |
| RSA | Rivest Shamir Adleman – An asymmetric cryptographic algorithm. |
| SHA | Secure Hash Algorithm. SHA-1 and SHA-512 are cryptographic hash functions designed by the National Security Agency (United States Department of Defence). |
| SOAP | Simple Object Access Protocol |
| TLS | Transport Layer Security |
| TSO | Transmission System Operator |
| URL | Uniform Resource Locator |
| UTF-8 | UCS (Universal Character Set) Transformation Format — 8-bit. |
| UUID | Universal Unique IDentifier |
| W3C | World Wide Web Consortium |
| WG | Working Group |

| | |
|--------------|---|
| WAN | Wide Area Network |
| XML | eXtended Markup Language |
| X.509 | An ITU-T standard for a Public Key Infrastructure (PKI) |

408 2.4 General overview

409 The purpose of the MADES standard is to specify a message delivery platform with the key
410 features shown in Figure 3.



411

412 **Figure 3 – MADES key features**

- 413 1. **Message delivery** – A party (sender) connected to the communication network can
414 send a message to another party (recipient), which is connected or can connect to the
415 network.
- 416 2. **Transparency** – Any transported message can be tracked down to gather trustworthy
417 information about the state of delivery and traversal path.
- 418 3. **Security** – Only the recipient of the message is capable of reading the message-
419 content. The sender of any message can be unambiguously verified.
- 420 4. **Reliability** – A message cannot get lost.
- 421 5. **Integration** – The MADES functions for sending and receiving messages can be
422 integrated with wide variety of technologies.
- 423 6. **Integrity** – MADES ensures that the content of a message has not been modified
424 during the delivery.

425 Note: The first four key features (message delivery, security, transparency and reliability) are
426 capabilities of the communication system, while the other one (integration) is a design
427 characteristic of the components of the communication system.

428 2.5 Message delivery and transparency

429 2.5.1 Message delivery

430 The main feature of MADES is the message delivery function, as shown Figure 4.

431 A message is transferred from a sender to a recipient. Both sender and recipient are business
432 applications (BAs). A BA connects to a MADES endpoint using a programming interface.

433 The sender and recipient view the MADES system only through the defined interface. The
434 document transported between sender and recipient can be any text or binary data. Alongside
435 with the document, a MADES message contains additional information, in a header (or
436 envelope), including information to securely identify, transport and route the message such as
437 a unique message ID, the identities of the sender and of the recipient, a business-type.



438

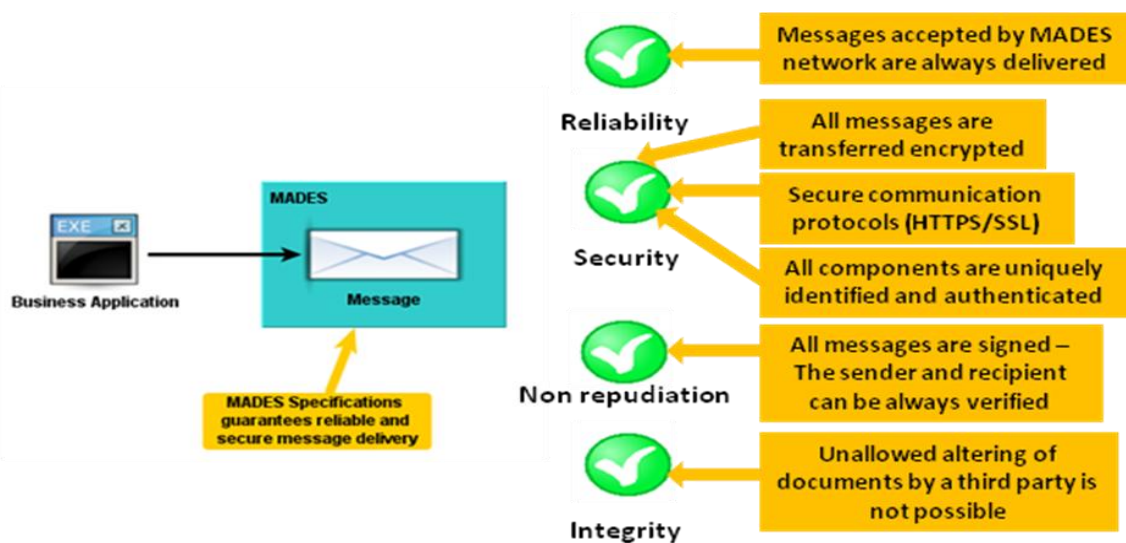
439 **Figure 4 – MADES message delivery overview**

440 **2.5.2 Transparency**

441 The message path — from the sender’s endpoint to the recipient’s endpoint — goes through
442 some components of the MADES network. When a message traverses a component, the later
443 notifies the event and a new message (referred as an acknowledgement) is sent back to the
444 sender’s endpoint. All the events notified during the message delivery can be retrieved by the
445 sender’s BA.

446 **2.6 Security and reliability**

447 MADES ensures a secure message transfer and a fully tracked delivery, as shown in Figure 5.



448

449 **Figure 5 – MADES security and reliability**

450 A MADES communication system guarantees that any message accepted by the system will
451 not be lost. The sender can at any time check the delivery status of the messages (delivering,
452 delivered or failed).

453 The standard describes a logging mechanism to be implemented in all message handling
454 components to provide information about the message transfers; MADES describes non-
455 repudiation features, allowing the verification of a message and its header which includes the
456 sender, the recipient, the sending time, the delivery time, etc.

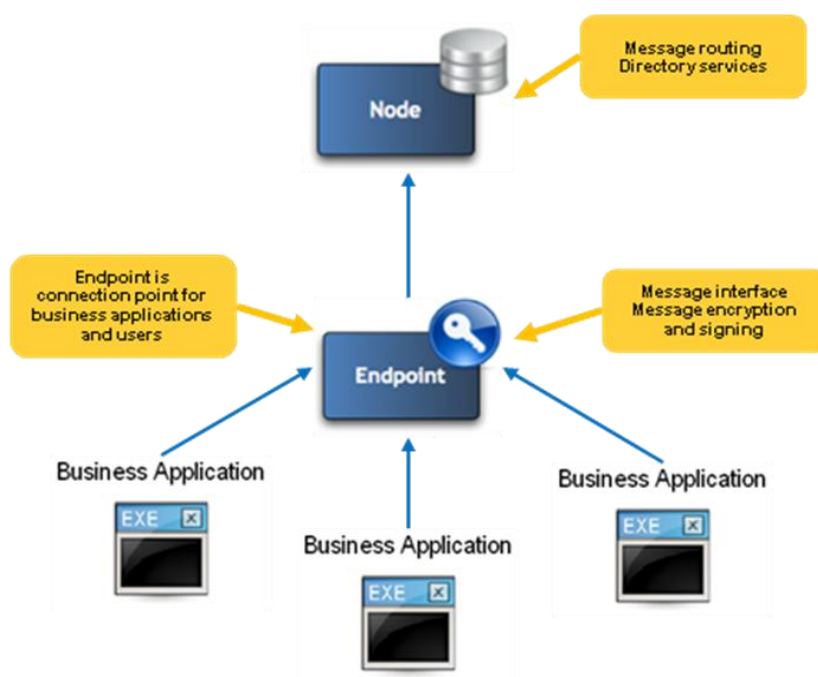
457 MADES defines the way to sign and encrypt the transported messages.

458 For the communication layer, the MADES components use the secure communication
459 protocols HTTPS. Information is transported encrypted. Moreover, both sides of
460 communication are authenticated using industry-standard PKI certificates.

461 The security features are detailed in § 2.10.

462 2.7 Main components

463 MADES describes two logical communication components and their interfaces, as shown in
464 Figure 6.



465

466 **Figure 6 – MADES components**

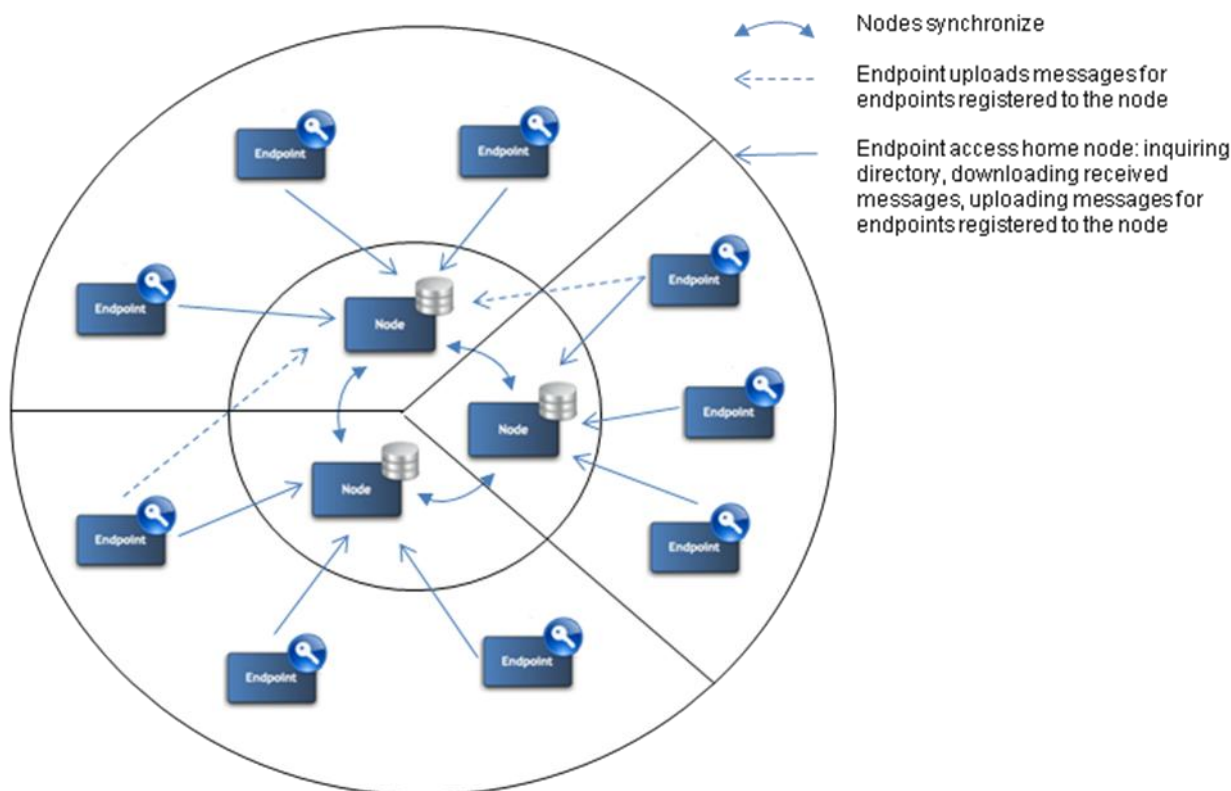
467 From the users' or business application (BA) point of view, the crucial component is the
468 endpoint, which provides the interface for the BAs to send and receive the messages. Actually
469 no graphical user interface is part of MADES; such an interface to provide a manual way for
470 sending and receiving messages is an application which can be integrated with the endpoint.

471 The node component serves as a central part of a MADES network. Each node contains a
472 directory with information on all the registered network components, whether endpoints or
473 nodes.

474 **2.8 Distributed architecture**

475 A MADES network may consist of multiple interconnected nodes, each taking care of a part of
476 the network, as shown in Figure 7.

477 A MADES network may contain a large number of collaborating components with the nodes in
478 the centre. A MADES network has a distributed architecture; it does not have any single
479 central component. All nodes have equal responsibilities; each manages a part of the
480 network.



481

482 **Figure 7 – MADES network distributed architecture**

483 Each endpoint shall register with a home node. The components registered with a node are
484 referred as the registered components. Endpoints currently connected to a node are the
485 connected endpoints.

486 Directory information about all registered endpoints is regularly shared between the nodes,
487 using the node synchronization interface; so that endpoints registered with different home
488 nodes can exchange messages.

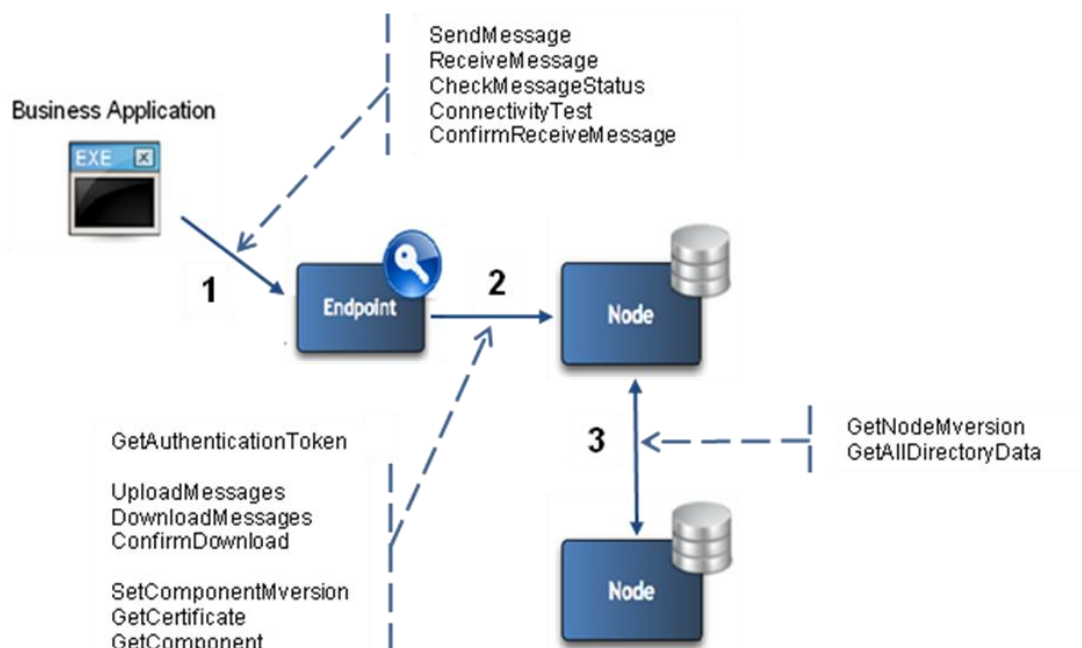
489 An endpoint can connect to any node to send messages, but it can only receive messages
490 from the home node.

491 **2.9 Components' exposed interfaces**

492 MADES standard specifies the interfaces between the components. All interfaces and
493 services are presented in Figure 8.

494 Each arrow on Figure 8 shows a component (at the tail of the arrow) using the interface
495 exposed by the component at the tip of the arrow.

- 496 1. Endpoint interface → used by a business application (BA) – see § 5.2.
 497 An endpoint shall implement this interface for a BA to connect to a MADES network.
 498 2. Node interface → used by the endpoints – see § 5.3.
 499 A node shall implement this interface to allow an endpoint to transfer the messages
 500 and to query the node directory.
 501 3. Node synchronization interface → used by the nodes – see § 5.3.5.
 502 A node shall implement this interface to synchronize directory data with the other
 503 nodes of the MADES network.



504

505 **Figure 8 – MADES interfaces and services**

506 **2.10 Security features**

507 **2.10.1 Overview**

508 Main goals of the MADES security definition are summarized by the following points:

- 509 • The security solution is transparent to the business applications (BAs) – no specific
 510 implementation shall be required in the application to communicate securely.
 511 • Any message shall be readable only by the recipient.
 512 • The sender of any message shall be unambiguously identified.
 513 • Non-repudiation of the messages – it shall be possible to unambiguously prove that the
 514 sender sent the message and that the recipient received it.
 515 • MADES ensures that the content of a message is not altered during the delivery process.
 516 • All communication routes shall be encrypted. The security solution complies with the
 517 X.509 public key infrastructure.

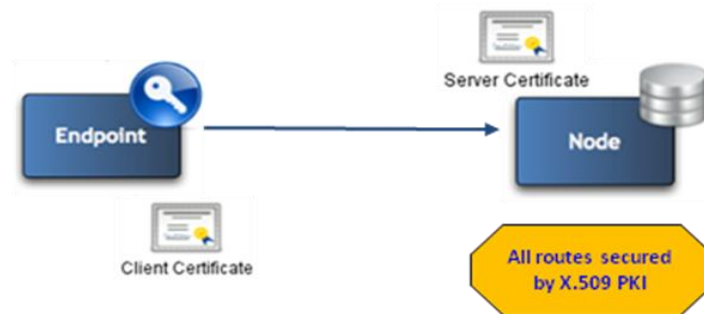
518 The security issues are covered on two levels: transport-layer security and message-level
 519 security.

520 On the transport-layer, MADES requires the communication between two components to
 521 always be encrypted. Two components that exchange information shall first unambiguously
 522 identify each other.

523 On the message-level, MADES requires that all messages shall be signed and encrypted, so
524 the sender of the message can be unambiguously identified and the message is only readable
525 by the intended recipient.

526 **2.10.2 Transport-layer security**

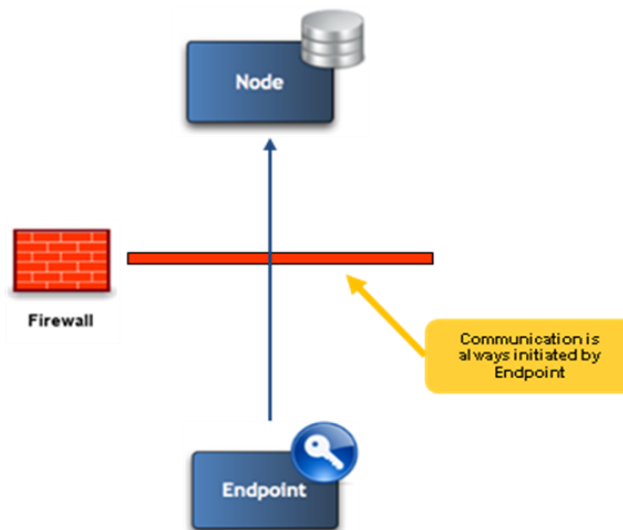
527 The transport-layer security of the communication between components relies on the
528 transport protocol layer. When communicating, components shall use a secure protocol
529 HTTPS providing the encryption of the communication route. Mutual authentication of
530 communicating components shall be handled using X.509 certificates; both the client and the
531 server shall authenticate themselves by their respective authentication X.509 certificates, as
532 shown in Figure 9:



533

534 **Figure 9 – MADES transport security overview**

535 The communication (i.e. the IP connection) between components shall always be initiated by
536 the client. This provides higher security on client-side by not having to allow incoming
537 connections through firewalls, as shown in Figure 10.



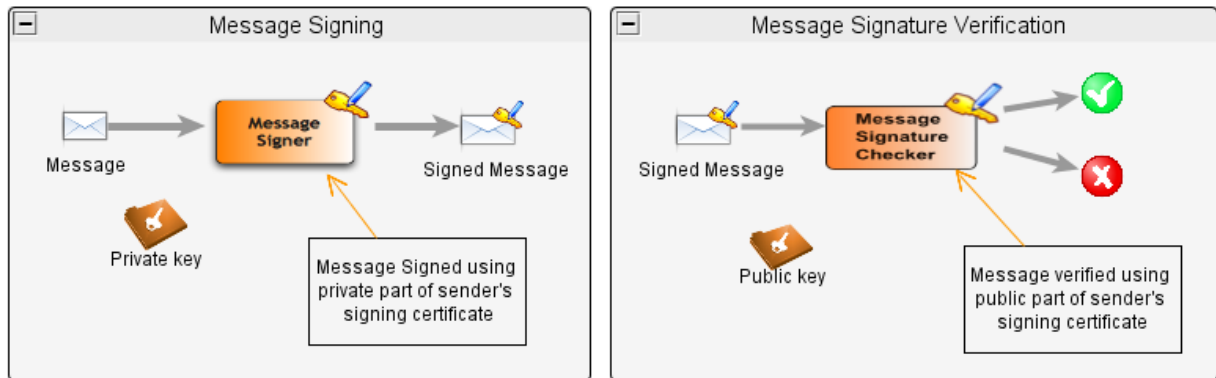
538

539 **Figure 10 – MADES secure communication initiation**

540 **2.10.3 Message-level security**

541 The unambiguous identification of the sender of any message sent via the MADES network is
542 enabled by usage of digital signatures as shown in Figure 11.

543 On the sender's endpoint, the message is signed using the sender's private key of a signing
544 certificate. On the recipient's endpoint, the message signature is verified using the sender's
545 public key of the certificate.

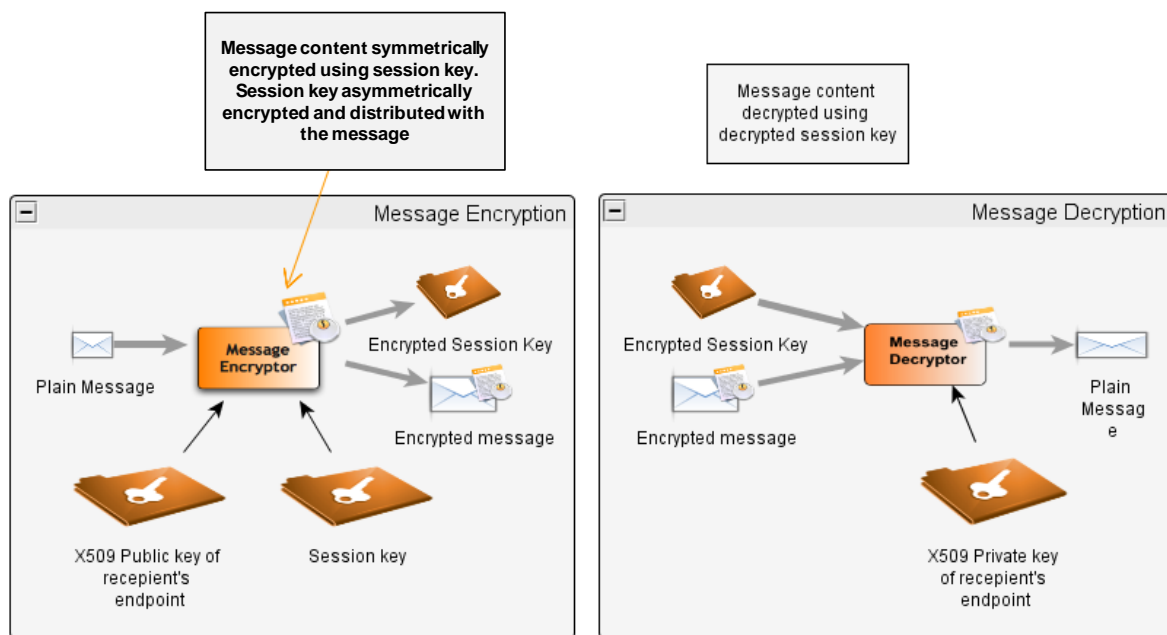


546

547

Figure 11 – Message signature

548 Any message sent via the MADES shall be encrypted, so that only the intended recipient can
549 read the message-content as shown in Figure 12:



550

551

Figure 12 – Message encryption and decryption

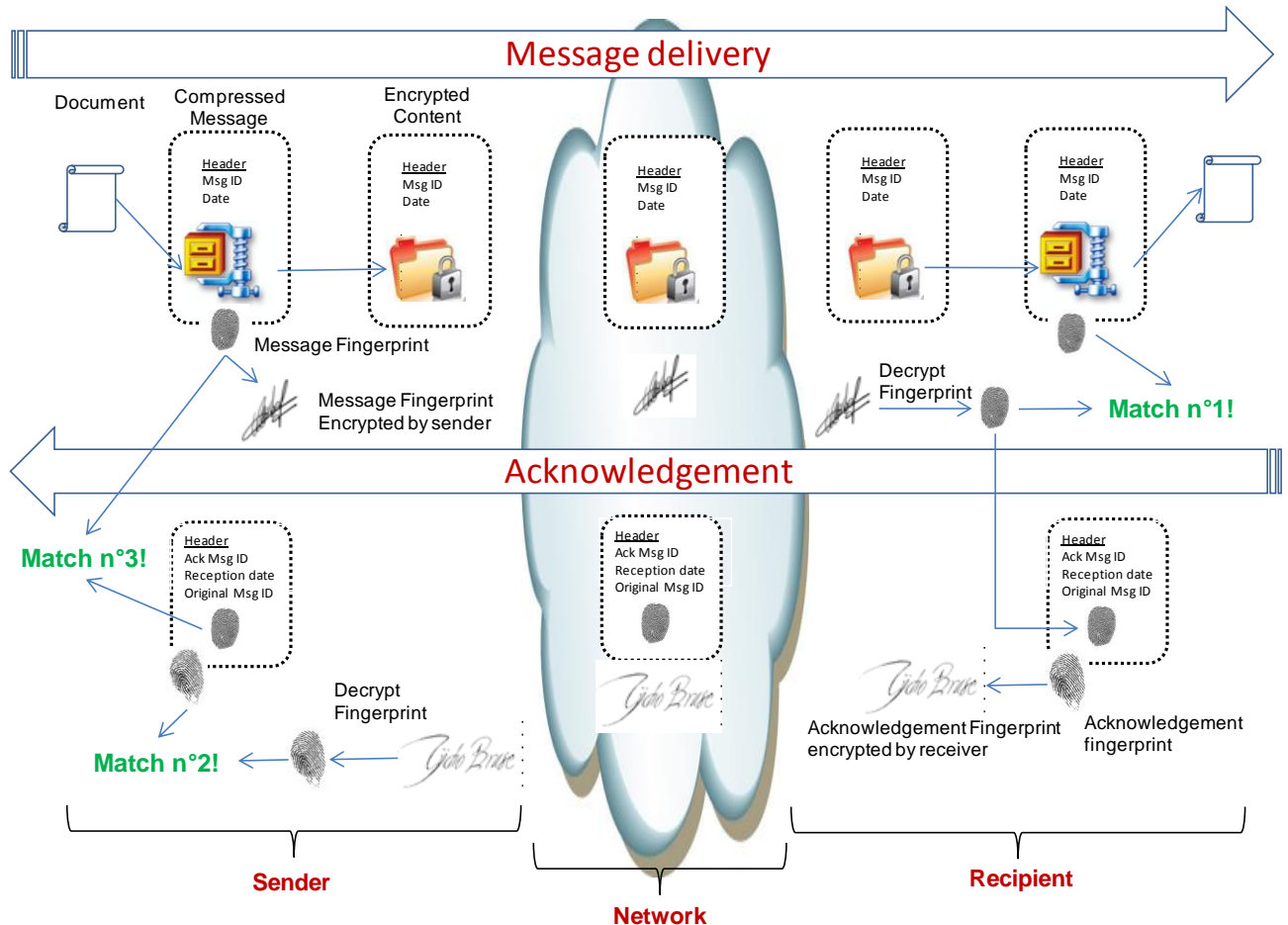
552 The content of the message (i.e. the document) is encoded with a randomly generated
553 session key, which is then itself encoded with the public key of the encryption certificate of
554 the recipient. The encoded key is transported together with the message.

555 The receiver decodes the key with the private key of his encryption certificate, and then uses
556 the key to decode the document.

557 **2.10.4 Non repudiation**

558 **2.10.4.1 Overview**

559 The general behaviour is first presented in Figure 13, and then detailed in § 2.10.4.2 and
560 § 2.10.4.3.



561

562

Figure 13 – Non repudiation

563 **2.10.4.2 Message delivery**

564 The message fingerprint identifies uniquely the document together with some header
565 information, such as the message unique ID (MsgID) and the sending date and time. The
566 document may have been previously compressed.

567 Both the fingerprint and the document are encoded and transported to the recipient. The
568 fingerprint is encoded in a way that uniquely identifies the sender (signature), and the
569 document in a way that only the recipient can read it (encryption).

570 The recipient decodes both the fingerprint and the document. He verifies (match n°1) that the
571 fingerprint, which he can regenerate from the message, matches the transported signed
572 fingerprint (signature verification). Then he stores the decoded message and the encoded
573 fingerprint. Both elements together with the signing certificate prove that the message was
574 sent by the sender.

575 2.10.4.3 Acknowledgement

576 The recipient sends back a new message, the acknowledgement, using a similar process. The
577 new message contains the unique ID of the original message (Original Msg ID) and the
578 attached document is the fingerprint of the original message¹.

579 The acknowledgement is signed but not encrypted and transported to the sender of the
580 original document.

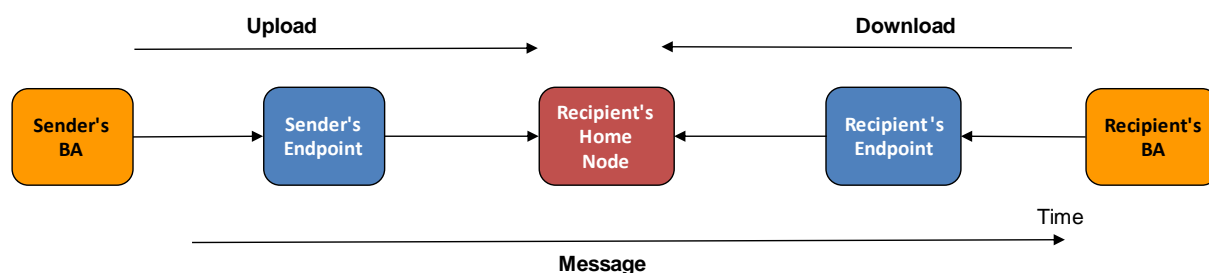
581 When he receives the acknowledgement, the sender verifies (match n°2) that the
582 acknowledgement was sent by the recipient (signature verification). He also verifies that the
583 acknowledgement document is the original message fingerprint (match n°3).

584 The set, composed of the original message, the signed acknowledgement and the signing
585 certificate, proves that the recipient received the original message.

586 3 Components' functions

587 3.1 Routing messages

588 A message shall be transported from the sender's BA (business application) to the recipient's
589 BA using the route shown in Figure 14.



590

591 **Figure 14 – Delivery route of a business-message**

592 The message is composed by the sender's endpoint with information and document provided
593 by the sender's BA. Then the message is transferred from component to component (from left
594 to right in Figure 14) until the recipient's BA.

595 The message-content (also referred as the message-payload) is the document provided by
596 the sender's BA. The composed message contains additional information (i.e. a header) used
597 for security, routing and delivery tracking.

598 The arrows in Figure 14 represent the IP connections between the components and each
599 arrow goes from a client to a server. Thus, a message is uploaded (or sent, or going out) on
600 the way from sender's BA to the recipient's node. It is downloaded (or received, or coming in)
601 on the way from the node to the recipient's BA. "Transfer" is the generic word used to mean
602 either upload or download.

603 The node, which the message goes through, shall be the home node of the recipient's
604 endpoint. Note that a reverse going message between the two BAs will not use the reverse
605 route if sender and recipient have different home nodes.

¹ There are several acknowledgements during the message delivery; the one referred to here is sent by the endpoint of the recipient party after it correctly receives the message (event n°4 in Figure 19).

606 There are two types of messages:

- 607 • Business-message – is a message composed by the sender's endpoint from a send
608 request initiated by a sender's BA. The goal of MADES is to transport such business-
609 messages to the requested recipient's endpoint.
- 610 • Acknowledgement – is an ancillary message used for tracking the end-to-end delivery
611 process of a business-message. The BAs do not know about those internal
612 acknowledgements, but a BA can request the endpoint about the delivery status of a
613 previously sent business-message.

614 3.2 Component and message unique identification (ID)

615 Each component shall have a unique ID in a MADES network. The identification scheme is a
616 network governance issue.

- 617 • The “component ID” (also referred as “component code”) is used to identify the component
618 when exchanging with other components.
- 619 • A BA shall use an endpoint ID to identify the recipient when sending a document.
620 Conversely the sender endpoint ID is provided to BA together with a received document.
- 621 • The IDs of the sender and the recipient are included in the header of each message.

622 When a component composes a message, it shall identify it with a UUID (Universal Unique
623 Identifier) — as defined in IETF RFC 4122 (<http://www.ietf.org/rfc/rfc4122.txt>).

624 NOTE When delivering a document, a recipient's endpoint supplies the BA with a guaranteed (i.e. authenticated)
625 sender's identity: the component ID of the sender's endpoint. However the sender's identity is also often included
626 within the document itself, and it is up to the BA that analyses the document to check that both identities match.

627 3.3 Business-type of a business-message

628 A MADES network may support multiple and concurrent business processes.

629 A party “P” having an endpoint connected to the network can operate several BAs,
630 implementing functions to support internal activities and exchanges with others parties in
631 accordance to the roles he plays in the business processes.

632 The BAs request the endpoint to send documents to other parties. The endpoint supports
633 concurrent requests from the BAs.

634 Other parties, while fulfilling their own roles in these business processes, may also send
635 some documents to party “P”. For dispatching correctly the received documents between BAs,
636 each BA may indicate a business-type when requesting for downloading a possibly newly
637 received document.

638 So the business-type is mandatory text information provided by a sender's BA, included in the
639 header of the business-message, transported with the message to the recipient's endpoint,
640 and used by a recipient's BA to retrieve the only documents it shall process.

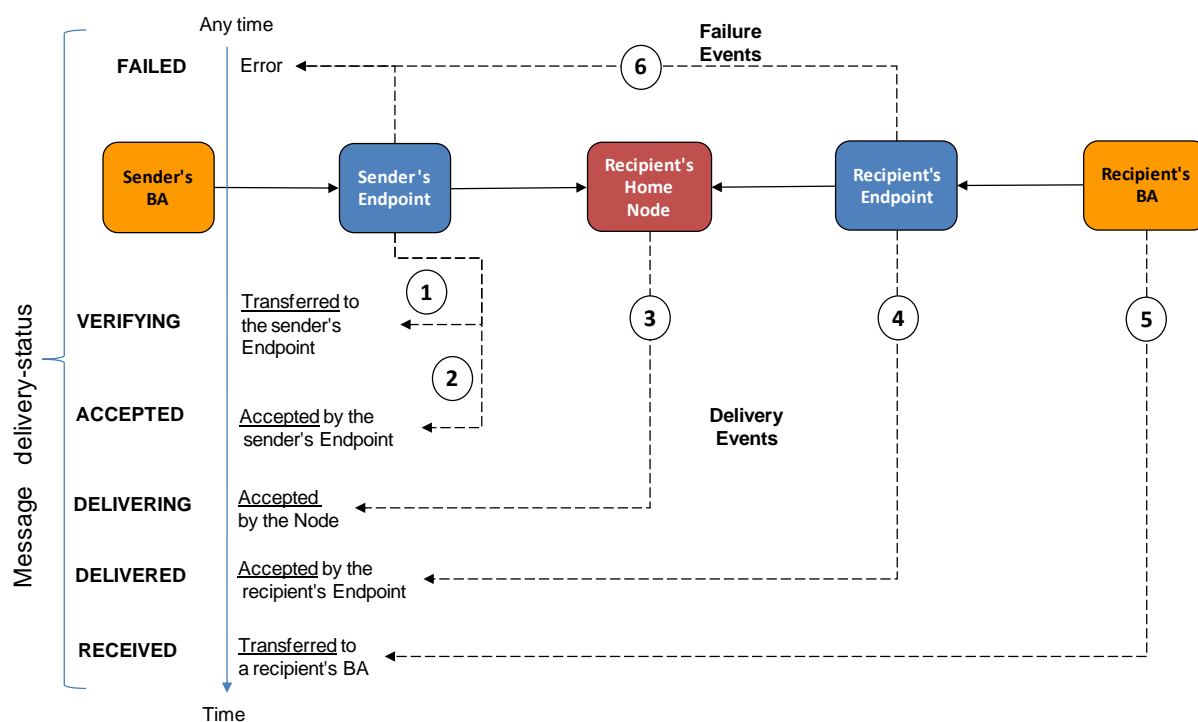
641 Each party is free to organise how he architectures activities, functions and BAs in his own
642 Information System, in a way transparent to the other parties. However the business-types
643 shall be agreed between all parties as part of the overall information exchange design².

² Business-types can be compared to port numbers: competing applications in a machine use different port numbers so that received information can be correctly routed. Used port numbers have to be agreed between parties (e.g. 21:FTP, 22:SSH, 25:SMTP) but they are not part of IP protocol which accepts any number.

644 **3.4 Delivery-status of a business-message**

645 The delivery process of each business-message is fully tracked. Tracking means that the
646 components taking part in the routing process notifies the sender's endpoint with events about
647 the message. The reported events are:

- 648 • Delivery event — notifies that the business-message has been either:
 - 649 a) transferred to a component; i.e. the component confirms it received the business-
650 message (“Transfer confirmation” is defined in § 3.8).
 - 651 b) accepted by a component; the component confirms that it received the business-
652 message and that the message successfully passed validation. It also means that the
653 message is ready to be transferred to the next component on the route to the
654 recipient's endpoint (“Acceptance” is defined in § 3.9)
- 655 • Failure event — notifies that a business-message cannot be delivered because:
 - 656 a) the message was rejected, for it fails the validation;
 - 657 b) or an unrecoverable error occurred when processing the message.



658
659 **Figure 15 – Reported events during the delivery of a business-message**

660 Figure 15 shows the possible events and the components that issue them.

661 The delivery-status of a business-message expresses the knowledge of the sender's endpoint
662 about the message delivery. The status can be requested at any moment by a sender's BA
663 providing the message ID returned by the sender's endpoint when the message was sent. The
664 possible values are provided in Table 1:

665

Table 1 – Message delivery status

| Message delivery-status | Notified event |
|-------------------------|---|
| VERIFYING | The business-message has been <u>transferred</u> to the sender's endpoint. Some additional checks are in progress before the endpoint may accept it, e.g.: the endpoint is waiting for signature by an external signing device (see § 3.16.3), or is waiting for the encryption certificate requested to the node directory. |
| ACCEPTED | The business-message has been <u>accepted</u> by the sender's endpoint. Conditions are met to transport the message in the network. |
| DELIVERING | The business-message has been <u>accepted</u> by the node. |
| DELIVERED | The business-message has been <u>accepted</u> by the recipient's endpoint. |
| RECEIVED | The business-message has been <u>transferred</u> to a recipient's BA. |
| FAILED | The message delivery has <u>failed</u> . So, the business-message will not be transported any further. |

666 The acknowledgement, which notifies that a message has been accepted by the recipient's
667 endpoint or transferred to a recipient's BA, does not and shall not mean more that the
668 document (i.e. the content of the message) has been technically and securely delivered to the
669 endpoint or a BA of the Information System (IS) of the recipient party. So far, the content of
670 the document has not been analysed. The probable and further analysis may result in a
671 "functional acknowledgment", the document being then accepted or rejected according to the
672 business rules. Such a functional acknowledgement can even be a new document that the
673 MADES network will be entrusted to deliver as a new business-message to the sender of the
674 original document.

675 3.5 Communication between components

676 3.5.1 Principle

677 To communicate, a client component establishes a secured communication channel with a
678 server component, and then issues requests through the channel.

679 The server component validates the request and replies. The client component receives back
680 a request status and validates the reply.

681 3.5.2 Establishing a secured communication channel between two components

682 A request from a client component to a server component shall only be processed after the
683 client has established a secured (i.e. encrypted) communication channel with the server.

684 The communication channel shall be secured using the HTTPS (HTTP over TLS) protocol. So
685 each peer, either client or server, verifies that the other peer is a valid and trusted network
686 component — see § 3.16.6.

687 A client component shall be able to connect to a server component through a network proxy.

688 An endpoint administrator shall be able to configure a primary and a secondary URL to
689 connect to the home node.

690 An endpoint may connect to any node for uploading business-messages and
691 acknowledgements addressed to a recipient's endpoint registered with the connected node.

692 The endpoint shall request by its home node directory the “routing information” for
693 establishing connection — see § 5.3.4.3.

694 The node URLs (primary and possibly secondary) in directory should rather contain FQDNs
695 (Fully Qualified Domain Names) than IP addresses to ease integration with the network
696 architecture constraints of the parties.

697 Concerning primary and secondary URLs: the nodes are key components and thus require
698 high availability. Availability techniques may vary, and redundancy or switch-over mechanism
699 may not be seamless to other components. So a node administrator may provide two URLs to
700 access his node. Consequently, the components that connect to a node shall implement a
701 mechanism to dynamically select the one URL which gains effective access.

702 **3.5.3 Token authentication of the client component**

703 Apart for the node-node synchronization, the server shall always first identify the client, i.e.
704 know its component ID to authorise the requests.

705 To do so, the client shall request the server for an authentication-token providing its own
706 component ID — see § 5.3.2.

707 The server provides back a token which:

- 708 • is a randomly generated string (e.g. a UUID).
- 709 • has a limited duration validity returned to the client. The later has to request for a new
710 token when expired or before the expiration time.

711 For every subsequent request (e.g. message transfer, directory query), the client shall always
712 provide the server with:

- 713 • the authentication-token;
- 714 • the signed authentication-token — “signed” means that the hash of the token is encoded
715 using the RSA algorithm — see § 3.16.1;
- 716 • the ID of the authentication certificate used for signing the authentication-token.

717 For every received request, the server shall process the following checks:

- 718 • the authentication-token is a known and not expired token;
- 719 • the certificate used to sign is a valid and non-revoked certificate owned by the client —
720 see § 3.16.8;
- 721 • the signature of the authentication-token is correct.

722 NOTE Such a token-based client authentication is neither part of nor linked to the TLS authentication, and thus
723 not constrained by specifics of software products used for the implementation of the component (e.g. web servers,
724 applications servers).

725 **3.5.4 Request authorisation**

726 A node shall reject a request for downloading messages or a request on directory if the client
727 component is not one of its registered endpoints.

728 A node shall reject a transfer request (download or upload) when it is already and
729 concurrently processing the same request for the same client — see § 3.8.

730 **3.5.5 Request/Reply validation**

731 The server shall validate data of any request and the client shall check the status and validate
732 data of any reply.

733 Validation prevents for foreseeable errors to occur and shall include:

- 734 1) Check that all mandatory request/reply elements are set.
- 735 2) Checks that all set elements do not contain any illegal characters, have the expected
736 format and size, and have values in expected list or range.
- 737 3) Check that the combination values of elements forms a valid set.

738 In case the request (or the reply) is a message transfer, the validation by the target
739 component shall include any additional check to ensure that the transferred messages can be
740 durably stored — see § 3.6 (e.g. the size of the message-content does not exceed the
741 maximal allowed size).

742 **3.6 Storing messages in components**

743 A component shall contain an internal message-box where it durably stores messages.
744 Durable (or persistent) means that the message shall be recovered after a software crash or a
745 hardware failure, when the component restarts (reboot or switch to a backup component in a
746 redundant architecture).

747 The stored information about a message (either business-message or acknowledgement)
748 shall be: the content, the header.

749 The endpoints shall store the compressed (if requested – see § 3.14.2) but non-encrypted
750 content of the message. The stored header shall include the message signature.

751 Within the message-box, a message shall be associated with additional information only used
752 locally by the component:

- 753 • Transfer timestamp — set by the component when the message is created/stored in the
754 message box, and used for priority management — see § 3.13.
- 755 • State — see § 3.7 for possible values and lifecycle.
- 756 • Priority — see § 3.13.
- 757 • Receive timestamp (only used for a business-message), the time when the message was
758 accepted by the recipient's endpoint — set when processing the acknowledgements of the
759 message — see § 3.10.4

760 A component administrator shall be able to configure a purge strategy for each business-type.
761 A purge strategy indicates how the component manages a business-message that has
762 reached the final state (see § 3.7). Possible strategies should include:

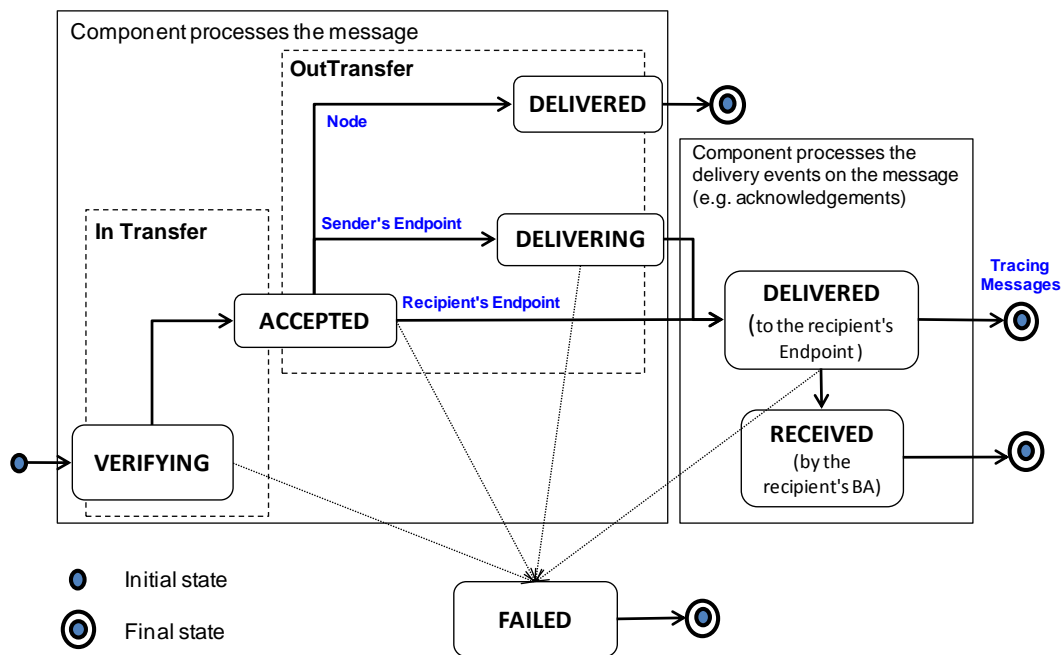
- 763 • Delete only the message-content (document).
- 764 • Delete the whole message and acknowledgements.
- 765 • Never delete the message.

766 A component administrator shall be able to (long term) archive messages and then delete the
767 correctly archived messages from the message-box.

768 **3.7 Lifecycle of a message state within a component**

769 A business-message has a local state in every component that processes it, and all these
770 states do not have the same values at the same time. The state is not transported data; it is

771 not part of the message header. The lifecycle of the state of a business-message within a
772 component is shown in Figure 16:



773

774 **Figure 16 – Lifecycle of the local state of a business-message within a component**

775 Possible states of a business-message are listed in Table 2:

776

Table 2 – Business message status

| Business-message State | Description |
|---------------------------------|---|
| Verifying | The successful transfer of the business-message in the component has been confirmed to the component which sent it and, before it may accept it, the message is currently passing some validation checks or pending (e.g. waiting for an external certificate to perform security operations such as signature/encryption). |
| Accepted | The business-message has been <u>accepted</u> by the component, and is pending for transfer to the next component on the message route. |
| Delivering | The business-message has been successfully <u>transferred</u> to the next component. |
| Delivered Received Failed | After the business-message has been transferred to the next component, the message state is set to the status of the acknowledgements coming back and which inform about the message delivery (see § 3.10). A business-message in the FAILED state is not delivered. A component shall set the business-message state to FAILED when it sends a failure-acknowledgement for the message. |

777 After a message has been successfully transferred (i.e. downloaded) from a node, the state
778 within the node shall move to DELIVERED which is the final state (and not to DELIVERING).
779 The reason is the following: If a node is not the home node of the sender's endpoint of a
780 business-message, no acknowledgement will ever inform about the rest of the message
781 delivery.

782 When a message is accepted by the recipient's endpoint, the state within the endpoint shall
783 be directly set to DELIVERED, because the message has reached the destination endpoint
784 and does not have a next component.

785 The delivery-status of a business-message, as defined in § 3.4, is the local state of the
786 message in the sender's endpoint.

787 **3.8 Transferring a message between two components (Handshake)**

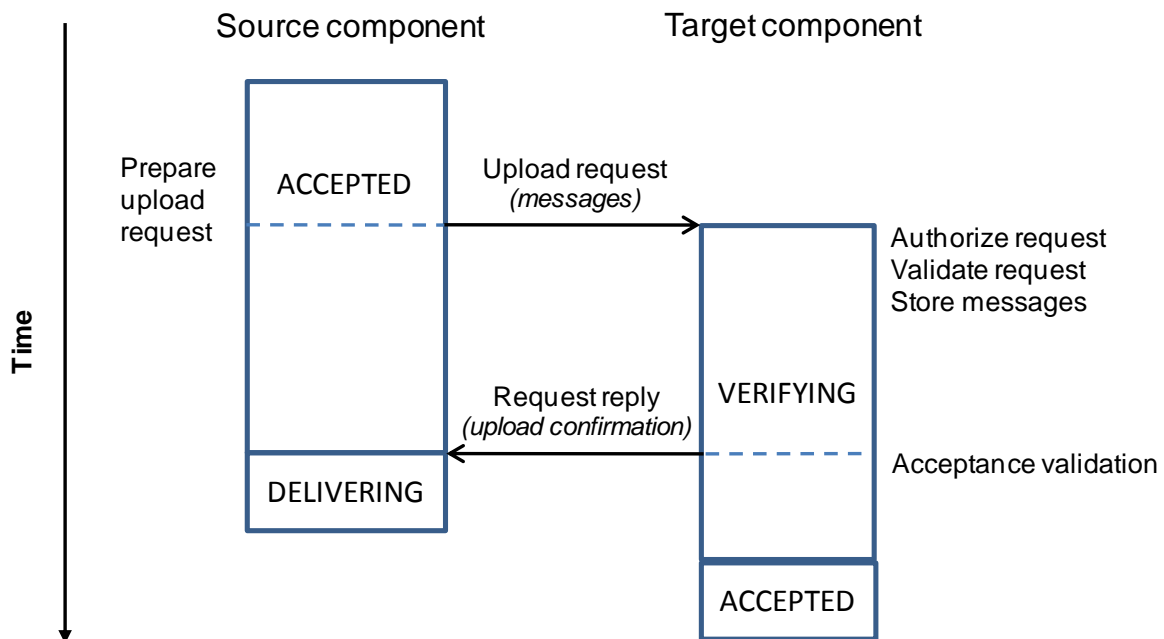
788 The transfer handshake is the mechanism which ensures that no message can be lost while
789 passing from a component to another. A component (referred as "target component") that
790 receives a message shall confirm to the sender component (referred as "source component")
791 that the message has been transferred.

792 A component is responsible for the message delivery from the moment it sends the transfer
793 confirmation to the previous component until the moment it receives the transfer confirmation
794 from the next component.

795 The target component confirms a message transfer to tell the source component that it took
796 responsibility for the message, and that it should not transfer it again. It means that either:

- 797 • the message has been stored in a durable way,
- 798 • the processing of the message has generated an error that has been logged (e.g.
799 message inconsistency).

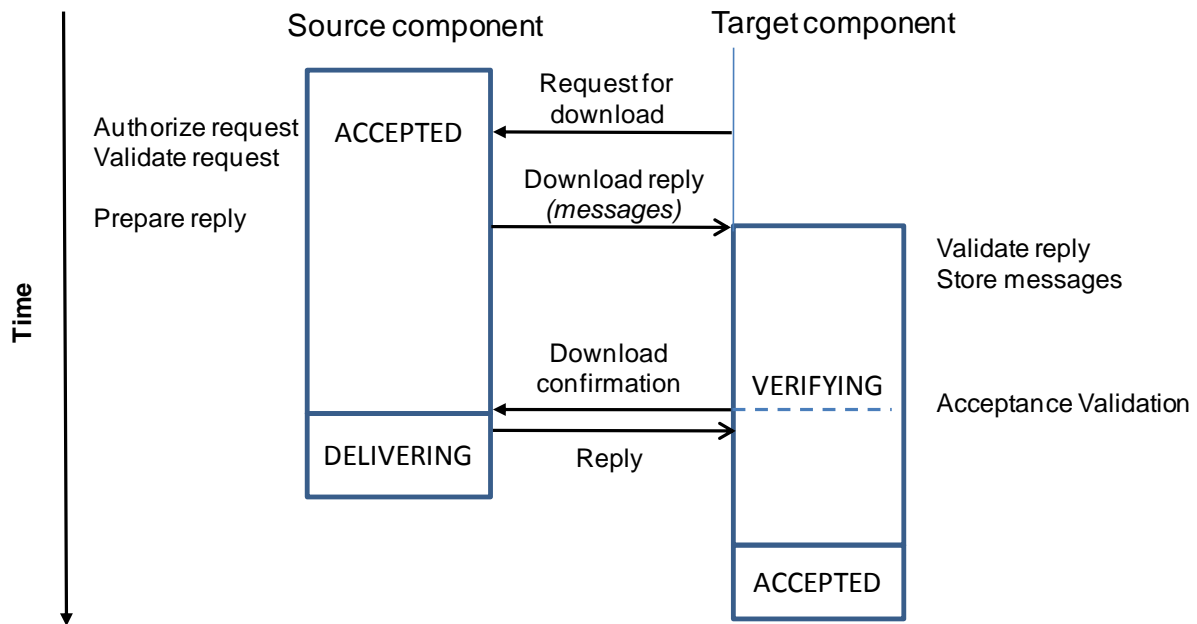
800 The handshake mechanism differs whether the transfer is an upload or a download, as shown
801 in Figure 17 and Figure 18. When downloading, the target component initiates the request
802 and an additional request is used to confirm the transfer to the source component.



803

804

Figure 17 – Transfer handshake when uploading of a message



805

806

Figure 18 – Transfer handshake when downloading of a message

807 NOTE The use of the VERIFYING state is a component internal design issue. A target component can confirm the
 808 transfer and set the message in the VERIFYING state before asynchronously processing the acceptance checks. A
 809 target component can confirm the transfer after it processes synchronously the acceptance checks, so the
 810 message state is directly set to ACCEPTED (or FAILED) — see § 3.9.

811 The handshake mechanism applies whether a transfer request contains one or several
 812 messages. Actually, the MADES interfaces for uploads and downloads can transfer bulk
 813 messages, mixing business-messages and acknowledgements — see § 5.3.3. When multiple
 814 messages are transferred simultaneously, the confirmation shall apply to all the transferred
 815 messages. Note that the BAs can only transfer (send or receive) business-messages one by
 816 one with their endpoint.

817 A server component shall not authorize a transfer request from a client component while the
 818 same request from the same client is currently being processed. This is necessary to fulfil the
 819 correct delivery sequence of two messages with the same business-type — see § 3.13. The
 820 bulk transfer is intended to gain performance without the use of concurrent requests.

821 The source component shall change the message state to the next state (generally
 822 DELIVERING) after it receives the transfer confirmation.

823 When the connection between the components is established or recovered after a failure, the
 824 source component shall transfer all pending messages in the ACCEPTED state. Note that it
 825 may happen that some of those messages have already been transferred, that the target
 826 component already sent the confirmation, but that the source component did not receive it or
 827 failed while processing it. So the target component may receive an already existing message
 828 (recognized with the message ID). In this situation, it shall then just confirm the transfer and
 829 log this duplicate transfer event.

830 **3.9 Accepting a message**

831 A component shall accept a transferred message after it passed the validation checks
 832 described in Table 3.

833

Table 3 – Accepting a message – Validation checks

| Component | Validation checks |
|----------------------|---|
| Sender's endpoint | <p>The transferred message can only be a business-message:</p> <ul style="list-style-type: none"> • Existence of the recipient's endpoint. • Availability of the encryption certificate of the recipient's endpoint, i.e. successfully retrieved from directory cache or from home node directory. • Successful signature of the message. <p>NOTE The business-message shall be compressed (if requested) while received by the endpoint, and it shall be encrypted when uploaded to the node.</p> |
| Node | <p>The transferred message can be a business-message or an acknowledgement:</p> <ul style="list-style-type: none"> • The recipient's endpoint has registered with the node. • The sender's endpoint exists in the directory and owns the certificate used to sign the message. • The certificates used for signing and encryption (if encrypted) exists and are not <u>revoked</u> (see § 3.16.8). |
| Recipient's endpoint | <p>The transferred message can be a business-message or an acknowledgement:</p> <ul style="list-style-type: none"> • Successful decryption of the content, when encrypted. • Successful verification of the signature, when signed. • When the message is an acknowledgement notifying the event n°4 (see Figure 19), successful match between the acknowledgement content and the original message fingerprint (hash). <p>NOTE A compressed business-message shall be uncompressed when transferring to a recipient's BA.</p> |

834 When a business-message is accepted, the following operations shall be processed as a
835 transaction³:

- 836 • The message is updated (e.g. decrypted content; change of the local state according to
837 the lifecycle).
- 838 • The component notifies the related delivery event – see § 3.10.2.

839 When a business-message is rejected, the component shall notify a failure-event and update
840 the message as a transaction.

841 When an acknowledgement is rejected, the component shall log the error and set the
842 acknowledgement state to FAILED; this stops the delivery.

843 3.10 Event management

844 3.10.1 Acknowledgements

845 A component can notify an event which occurs when delivering a business-message, by
846 sending an acknowledgment to the sender's endpoint of the message. The business-message
847 on which the event occurs is referred as the original message, and its ID shall be included in
848 the acknowledgement header.

849 An acknowledgement shall be routed and delivered using the same transfer (upload and
850 download) mechanism as the business-messages, without being acknowledged itself.

³ In the whole specification a "transaction" means an operation that shall succeed or fail as a complete unit and cannot remain in an intermediate state.

851 An acknowledgement shall have the same business-type as the original message.

852 The content of an acknowledgement shall never be compressed or encrypted.

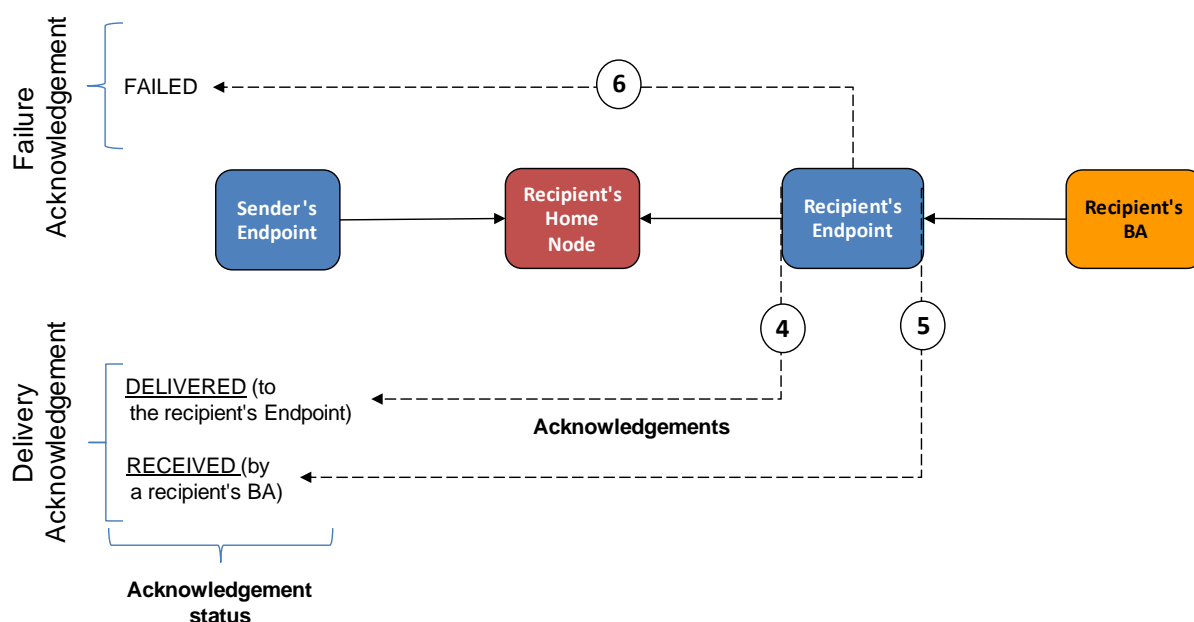
853 **3.10.2 Notifying events**

854 “Notifying an event” on a message means either:

- 855 • Sending an acknowledgement containing event information to deliver to the sender’s endpoint of the message.
- 856
- 857 • Except when the event is notified by the sender’s endpoint of the message itself, then the
- 858 event information is just locally stored.

859 The event issuer shall update the message according the event (e.g. the message state).

860 Previous operations shall be realized as a transaction, and the issuer shall log the event.



861

862 **Figure 19 – Acknowledgements along the route of the business-message**

863 Figure 19 shows the issuers and the events characteristics notified by acknowledgements.
 864 Table 4 provides the event characteristics where the events are numbered as in Figure 19.

865 **Table 4 – Characteristics of notified events**

| Event | Event characteristics |
|-------|--|
| 1 | <p><u>Status:</u> VERIFYING</p> <p><u>Issuer:</u> Sender’s endpoint</p> <p><u>Acknowledger:</u> None (The event is internal to the Sender’s endpoint and does not generate an acknowledgement)</p> |
| 2 | <p><u>Status:</u> ACCEPTED</p> <p><u>Issuer:</u> Sender’s endpoint</p> <p><u>Acknowledger:</u> None (The event is internal to the Sender’s endpoint and does not generate an acknowledgement)</p> |

| Event | Event characteristics |
|-------|---|
| 3 | <p><u>Status</u>: TRANSPORTED</p> <p><u>Issuer</u>: Recipient's node</p> <p><u>Acknowledger</u>: None (Although the event is coming from the node, it is notified by the sender's endpoint and does not generate an acknowledgement)</p> <p><u>Comment</u>: A node never sends an acknowledgement because it could not be delivered if it is not the home node of the sender's endpoint of the original message. The reason is that the sender's endpoint will never connect for downloading messages, including acknowledgements. Thus the node delegates to the sender's endpoint the issuance of the acknowledgement by notifying in the upload response whether it accepts or rejects the business-message.</p> |
| 4 | <p><u>Status</u>: DELIVERED</p> <p><u>Issuer</u>: Recipient's endpoint</p> <p><u>Acknowledger</u>: Recipient's endpoint</p> <ul style="list-style-type: none"> ✓ Content: the non-encoded message fingerprint (hash) of the original message — see § 3.14.3. ✓ Internal type: DELIVERY_ACKNOWLEDGEMENT ✓ Signed: Yes ✓ Original message state: DELIVERED <p><u>Comment</u>: State and status are set to DELIVERED because the acknowledger is the recipient's endpoint of the original message.</p> |
| 5 | <p><u>Status</u>: RECEIVED</p> <p><u>Issuer</u>: a recipient's BA</p> <p><u>Acknowledger</u>: Recipient's endpoint</p> <ul style="list-style-type: none"> ✓ Content: irrelevant but as least one character. ✓ Internal type: RECEIVE_ACKNOWLEDGEMENT ✓ Signed: No ✓ Original message state: RECEIVED <p><u>Comment</u>: A recipient's BA (not a MADES component) delegates to the recipient's endpoint the issuance of the acknowledgement notifying that the successful transfer of the business-message.</p> |
| 6 | <p><u>Status</u>: FAILED</p> <p><u>Issuer</u>: any component.</p> <p><u>Acknowledger</u>: if not notified by the sender's endpoint</p> <ul style="list-style-type: none"> ✓ Content: an English readable description of the encountered error or the reason why the message was not accepted. ✓ Internal type: FAILURE_ACKNOWLEDGEMENT ✓ Signed: No. ✓ Original message state: FAILED <p><u>Comment</u>: The event is referred as the "failure-event". In case a failure-event occurred in the sender's endpoint it processes it internally and does not send an acknowledgement.</p> |

866 The meaning of the characteristics is provided in Table 5.

867 **Table 5 – Event characteristics description**

| Characteristic | Description |
|----------------|--|
| Status | The value to set to the "state" element of the "trace" item (see Table 70) reporting the event to the sender's BA through the <i>CheckMessageStatus</i> service - see § 5.2.2.3. |
| Issuer | The component that notifies (issues) the event. |

| Characteristic | Description |
|------------------------|---|
| Acknowledger | The component that sends the acknowledgement, possibly none or possibly different from the issuer. |
| Content | The content of the acknowledgement message. |
| Signed | Whether the acknowledgement message is signed or not. |
| Internal type | The value to assign to the <i>internalType</i> element of the acknowledgement – see Table 61 |
| Original message state | The value to set to the local state of the original message in the issuer component when issuing the event. |

868 3.10.3 Lifecycle of an acknowledgement

869 Table 6 provides the possible values for the local state of an acknowledgement within a
870 component; these are a subset of the states of a business-message.

871 Unless signed using an external device (see § 3.16.3), an acknowledgement is created in the
872 ACCEPTED state, otherwise in the VERIFYING state.

873 **Table 6 – Acknowledgement state description**

| Acknowledgement State | Description |
|-----------------------|--|
| Verifying | The acknowledgement has been created by the component or successfully transferred to it, and a signature operation is currently processed or pending (e.g. waiting for the external device to be signed, or waiting for the external certificate to verify the signature). |
| Accepted | The acknowledgement is pending for transfer to the next component towards the destination endpoint. |
| Delivered | The acknowledgement has been successfully transferred to the next component or has reached the destination, i.e. the sender's endpoint of the original message. |
| Failed | The component encountered an unrecoverable error when processing the acknowledgement. The acknowledgement will never be transferred to another component. |

874 3.10.4 Processing a transferred acknowledgement

875 A component shall always accept a transferred acknowledgement. When processing a
876 transferred acknowledgement:

- 877 • The component shall log the event notified by the acknowledgement.
- 878 • In case an unrecoverable or an acceptance error (see § 3.9) occurs, the component shall
879 set the acknowledgement and the original message to the FAILED state in a transactional
880 way, and log the error. This stops the acknowledgement delivery.
- 881 • Otherwise the component shall in a transactional way:
 - 882 – update the state of the original message according to the event in conformance to
883 Figure 15;
 - 884 – when the event is n°4 (see Figure 19), set the “receive timestamp” of the original
885 message to the time the acknowledgement was created (generated item –
886 see Table 61).

887 The original message may not exist in a node for it was delivered through another node. The
888 acknowledgement shall then be correctly processed and route.

889 The original message may be in the ACCEPTED state. This may happen when the message
890 was transferred and the component did not receive the confirmation. When the connection is
891 back, it may receive the acknowledgement before the message is transferred again.

892 **3.11 Message expiration**

893 **3.11.1 Principle**

894 The message expiration is a mechanism to notify the sender's BA that a business-message
895 has not been delivered in the due time to the recipient's endpoint. When the time limit is
896 exceeded, the sender's endpoint changes the state of the message to FAILED.

897 The expiration time of a business-message is the time limit when the sender's endpoint
898 declare that the message delivery has failed, because it has not received the
899 acknowledgement notifying that the message was accepted by the recipient's endpoint (event
900 n°4 in Figure 19).

901 **3.11.2 Setting the expiration time of a message:**

902 An endpoint administrator shall be able to configure maximum durations for the delivery of the
903 business-messages as:

- 904 • duration values associated to the business-types;
- 905 • a non zero and positive default duration value.

906 The expiration time is part of the header of a message — see Table 61, expirationTime. The
907 time count shall start when the sender's endpoint confirms the transfer of the business-
908 message (event n°1). The expiration time shall be set by the sender's endpoint according to
909 the business-type of the message. Otherwise the default duration value shall be used.

910 The expiration time of an acknowledgement is the expiration time of the original message.

911 **3.11.3 Looking for the expired messages:**

912 Each component shall cyclically look for the expired messages either business-messages or
913 acknowledgements. A message expires when the expiration time is past and the local state is
914 not amongst DELIVERED, RECEIVED or FAILED.

915 The sender's endpoint shall notify the expiration of a business-message using an event-
916 failure. Otherwise the component shall set the local message states to FAILED and log the
917 expiration (date, message ID, sender, recipient, sending time, expiration time).

918 NOTE The default value for maximum delivery duration is a general mechanism to set to FAILED the state of the
919 messages whose delivery cannot be processed for whatever reason, ensuring then that they will not be forever
920 delivering (i.e. "zombie" messages).

921 **3.12 Checking the connectivity between two endpoints (Tracing-messages)**

922 A tracing-message is a business-message used to check end-to-end connectivity between two
923 endpoints using the message tracking process. The message header contains a special type
924 for a tracing-message (TRACING_MESSAGE — see Table 62).

925 A BA can request to process a connectivity test with any endpoint. The sender's endpoint
926 shall then compose and send a tracing-message to deliver to the required destination
927 endpoint.

928 To check that the tracing-message reached the recipient's endpoint, the sender's BA can
929 check its delivery status, as for any business-message.

930 The business-type and the content of a tracing-message are irrelevant but shall have at least
931 one character. As any business-message, a tracing-message is signed and the content is
932 encrypted. So the tracing-message delivery success includes the checks of the certificates'
933 set-up and processing.

934 The header of the acknowledgements whose original messages are tracing-messages also
935 have a special type (TRACING_ACKNOWLEDGEMENT — see Table 62).

936 Because no recipient's BA will ever request for the tracing-message, the final state of a
937 tracing-message is DELIVERED in all components — see § 3.7.

938 **3.13 Ordering the messages (Priority)**

939 A component administrator shall be able to configure priority values according to the
940 business-types, and to configure a default priority value for unknown business-types.

941 A business-message shall have the priority configured for the business-type if defined;
942 otherwise the default priority.

943 The component shall process pending messages and elaborate the transferred list of
944 messages using the following order:

945 1) A message with higher priority is processed first.

946 2) If two messages have the same priority, the one that was first transferred by the
947 component is processed first — see "Transfer timestamp" in § 3.6.

948 The message priority is local to a component. It may differ between components and is not
949 transported information.

950 Assume that two messages (M1 and M2) of the same business-type are sent in this order by
951 BA1 to BA2. If BA2 receives both messages, M1 shall be received first. Whatever priority is
952 configured for the business-type by each component, the delivery order shall remain
953 unchanged.

954 An acknowledgement has the same priority as the original message, because it has the same
955 business-type.

956 The priority of the tracing-messages may be configurable; otherwise they have the default
957 priority.

958 **3.14 Endpoint**

959 **3.14.1 Endpoint functions**

960 An endpoint provides interfaces for BAs to send and receive messages in a secure way. An
961 endpoint shall provide the following functions:

962 • Communication:

963 a) Connect to a node using HTTPS.

964 b) Validate the send-requests from the BAs.

965 c) Validate the receive-requests from the BAs and provide the received documents.

966 • Pre-processing the to-send business-messages:

967 a) Compose the business-messages (e.g. create the message ID, set the expiration time,
968 and compress the content).

- 969 b) Check the existence of the recipients and get their encryption certificates by the home
970 node directory.
- 971 c) Generate the message signature and encrypt the message-content.
- 972 • Post-processing the received business-messages:
- 973 a) Get the signing certificates by the home node directory.
- 974 b) Decrypt the message-content, verify the message signature, and uncompress the
975 message-content.
- 976 • Notifying the events on the message delivery:
- 977 a) Send and process the acknowledgements.
- 978 b) Verify the signature and the content of the acknowledgements.
- 979 • Processing the messages:
- 980 a) Upload the encrypted business-messages to recipient's node.
- 981 b) Download the encrypted business-messages from the home node.
- 982 c) Store the messages in the local message-box.
- 983 d) Process the validation checks.
- 984 e) Look cyclically for the expired messages.
- 985 f) Update the messages' states according to delivery progress.
- 986 g) Manage the queues with messages pending for uploading or downloading.
- 987 h) Process the messages according to local priority rules.
- 988 • Processing the tracing-messages:
- 989 a) Validate the connectivity test requested by the BAs.
- 990 b) Compose the message.
- 991 c) Process the tracing-messages downloaded from the home node.
- 992 • Requesting the home node for directory information:
- 993 a) Retrieve other endpoints signing and encryption certificates.
- 994 b) Durably store the used signing certificates of the other endpoints.
- 995 • Replying to the messages status requests from BAs.
- 996 • Administration:
- 997 a) Synchronize the endpoint time with a reliable source (recommendation is to use a
998 standard OS mechanism such as NTP, the Network Time Protocol).
- 999 b) Install endpoint and CAs certificates (initial and renewed).
- 1000 c) Archive and purge the logs.
- 1001 d) Archive and purge the messages.

1002 3.14.2 Compression

1003 The sender's endpoint shall compress the content of each business-message whose
1004 business-type is configured to do so.

1005 The endpoint administrator shall be able to configure the business-types of the business-
1006 messages that shall be compressed.

1007 The tracing-messages and the acknowledgements shall not be compressed.

1008 Compression shall be done using the ZIP algorithm.

1009 Compression means that the message-content is encoded and that the metadata⁴ (see Table
1010 7) shall be added to the message header — see Table 61:

1011 **Table 7 – Compression – metadata attributes**

| Metadata Attribute Name | Metadata Type | Description |
|-------------------------|---------------|--|
| Compression | BOOLEAN | Value:= true (i.e. the message was compressed) |

1012 The header of a non-compressed message may also contain the metadata with the attribute
1013 value set to “false”.

1014 **3.14.3 Signing**

1015 The signing principles are presented in § 3.16.1.

1016 Only the endpoints sign the messages. A sender’s endpoint shall sign every business-
1017 message. The recipient’s endpoint shall sign the acknowledgement notifying event n°4 (see
1018 Figure 19).

1019 An endpoint shall verify the signature of every signed message that it receives. A message is
1020 signed if it contains the signature metadata (see Table 8).

1021 Signature algorithm shall be RSA-SHA (IETF RFC 3110 - <http://www.ietf.org/rfc/rfc3110.txt>).
1022 The signature format shall comply with the “XML Signature Syntax and Processing standard”
1023 (<http://www.w3.org/TR/2002/REC-xmlsig-core-20020212/>).

1024 An endpoint shall encode the message hash using the private key of the signing certificate.

1025 The manifest used to generate the message hash is:

1026 *Compress (content) + baMessageID + extension + generated + internalType + messageID*
1027 *+ relatedMessageID + receiverCode + senderCode + senderDescription +*
1028 *SenderApplication + businessType.*

1029 Where:

- 1030 • the italic names refer to the attributes of the internal message structure as described in
1031 Table 61;
- 1032 • “+” is the binary concatenation of the message attributes in UTF-8 encoding;
- 1033 • Note that the content of the message may be compressed or not, according to § 3.14.2,
1034 but not encrypted.

1035 Signing means that following metadata is added to the message header — see Table 61.

1036 **Table 8 – Signing – metadata attributes**

| Metadata Attribute Name | Metadata type | value |
|-------------------------|---------------|---|
| Algorithm | STRING | Value:= SHA-512 (The algorithm used to generate the message hash). |

⁴ « Metadata » refers to the part of the message header named “metadata” — see Table 61.

| Metadata Attribute Name | Metadata type | value |
|-------------------------|---------------|---|
| Certificate ID | STRING | The ID of the certificate whose private key was used to generate the signature, i.e. to encode the message hash — see § 3.16.2. |
| Signature | STRING | The message signature compliant with the “XML Signature Syntax and Processing standard”. The XML signature document is embedded here as a string. <i>(An example of an XML signature document is provided in § 5.6.4).</i> |

1037 When receiving a message an endpoint shall check the message was signed, i.e. if the
1038 header contains signature metadata, and then:

- 1039 1. Recover the signing certificate from the cache or from the home node.
- 1040 2. Verify that the certificate was valid when the message was generated (Certificate
1041 expiration date-time is a certificate attribute. The generated date-time of a message is
1042 part of the message header);
- 1043 3. Verify the XML signature (i.e. the “signature” attribute of the metadata) using the
1044 public key of the signing certificate.
- 1045 4. Regenerate the message hash of the received message using the “Algorithm”.
- 1046 5. Verify that the hashes provided by operations 3 and 4 are equal.

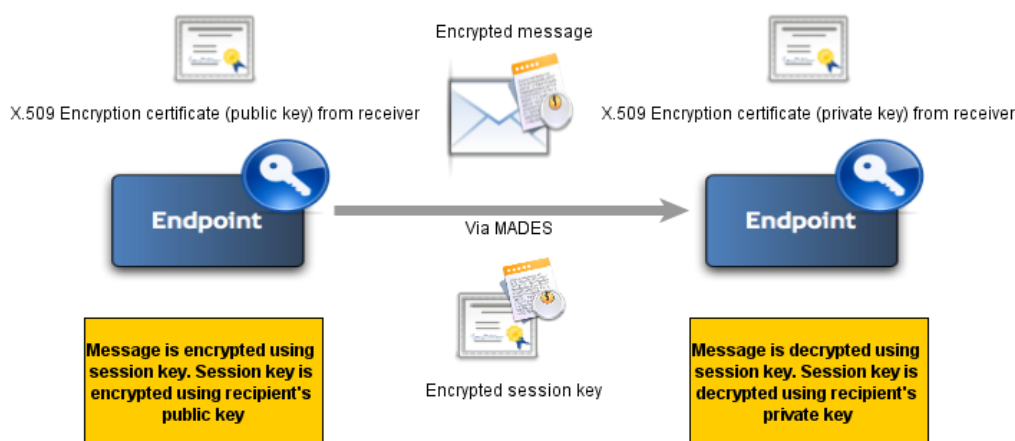
1047 3.14.4 Encryption

1048 The encryption principles are presented in § 3.16.1.

1049 The sender’s endpoint shall encrypt a business-message just before it is uploaded. The
1050 recipient’s endpoint shall decrypt a business-message just after it is downloaded.

1051 Only the message-content of the business-message shall be encrypted. The
1052 acknowledgments shall never be encrypted.

1053 The encryption and decryption processes use a combination of asymmetric and symmetric
1054 cryptography, as shown in Figure 20.



1055

1056 **Figure 20 – Encryption process**

1057 The sender’s endpoint shall use the encryption certificate of the recipient’s endpoint. It
1058 retrieves it by the home node.

1059 To encrypt the message-content, the endpoint shall first generate a random symmetric
1060 encryption key (called the session key), which is used to encode the content of the message.
1061 Then the symmetric key shall be encoded using the public key of the encryption certificate of
1062 the recipient's endpoint.

1063 The symmetric algorithm used to encode the message shall be AES (Advanced Encryption
1064 Standard) and the key size shall be 256 bits.

1065 Encryption means that the message-content is encoded and the metadata (see Table 9) shall
1066 be added to the message header — see Table 61):

1067 **Table 9 – Encryption – metadata attributes**

| Metadata Attribute Name | Metadata Type | Description |
|-------------------------|---------------|--|
| Cipher | STRING | Value:= AES-256 (The algorithm used to encrypt the message-content with the session key). |
| Certificate ID | STRING | The ID of the certificate whose public key is used to encode the session key — see § 3.16.2. |
| Session key | BYTE_ARRAY | The value of the session key encoded using RSA algorithm and the public key of the encryption certificate. |

1068 When receiving a message the recipient's endpoint shall check if the message is encrypted,
1069 i.e. if the header contains encryption metadata, and then:

- 1070 1. Verify that the certificate ID used to encrypt the message is one of the owned
1071 encryption certificates;
- 1072 2. Verify that the encryption certificate was valid when the message was generated
1073 (Certificate expiration date-time is a certificate attribute. The generated date-time of a
1074 message is part of the message header);
- 1075 3. Decode the symmetric session key using the corresponding private key of the
1076 certificate;
- 1077 4. Decode the message-content using the decoded session key and the algorithm used
1078 for encryption.

1079 3.15 Node

1080 3.15.1 Node functions

1081 A node shall provide the following functions:

- 1082 • Communication:
 - 1083 a) Authorize the HTTPS connections from the endpoints or from the other nodes.
 - 1084 b) Connect to the other nodes using HTTPS.
- 1085 • Processing the messages:
 - 1086 a) Upload and download the messages to and from the endpoints.
 - 1087 b) Store the messages in the local message-box.
 - 1088 c) Process the validation checks.
 - 1089 d) Look cyclically for the expired messages.
 - 1090 e) Update the messages' states according to the delivery progress.
 - 1091 f) Manage the queues of messages pending for downloading.
 - 1092 g) Process the messages according to the local priority rules.

- 1093 • Directory services:
- 1094 a) Provide the registered endpoints with the nodes' URLs and the endpoints' description
- 1095 and certificates.
- 1096 b) Request the others nodes for their reference directory data — see § 3.15.2.
- 1097 c) Reply to the others nodes' synchronization requests — see § 3.15.2.
- 1098 d) Manage directory data and the data version (Dversion).
- 1099 • Administration:
- 1100 a) Register the endpoints.
- 1101 b) Generate the certificates for the registered endpoints
- 1102 c) Import signing and encryption certificates from external CAs.
- 1103 d) Revoke the endpoint certificates — see § 3.16.8.
- 1104 e) Import the synchronization nodes' List — see § 3.15.3.
- 1105 f) Synchronize the node time with a reliable source (recommendation is to use a
- 1106 standard OS mechanism such as NTP, the Network Time Protocol).
- 1107 g) Archive and purge the logs.
- 1108 h) Archive and purge the messages.

1109 3.15.2 Synchronizing directory with other nodes

1110 A node directory is the master reference for all data regarding a sub-network composed of the

1111 node itself and the registered endpoints.

1112 The synchronization between the nodes is carried out cyclically or on the node administrator

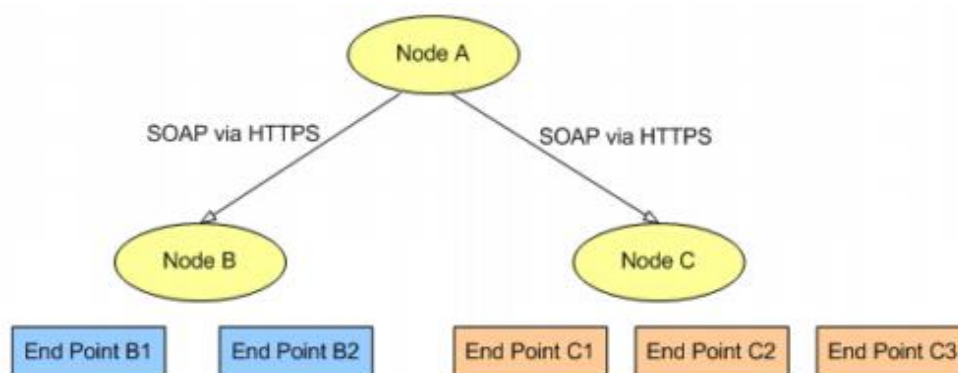
1113 demand. Each node requests the others nodes for their sub-network data and stores it in its

1114 own directory.

1115 The synchronization frequency is defined by the network governance.

1116 Example: Figure 21 shows the node A that connects to the node B, and then to the node C to

1117 obtain their directory data.



1118

1119 **Figure 21 – A node synchronizes with two other nodes**

1120 Each node shall manage a directory version number, referred as the “Dversion” for the

1121 reference data, which increases every time they are updated.

1122 Each node shall store reference data of the other nodes and the corresponding Dversion. The

1123 Dversion shall always be transferred together with the directory data. When requesting for

1124 data of another node, the client shall provide the Dversion of the remote data that it already
1125 possesses, so the reply can just inform that data is already up-to-date. Thus the nodes may
1126 synchronise frequently (e.g. 5 minutes).

1127 Synchronized data shall include:

- 1128 • information and certificates of all endpoints registered with the node,
- 1129 • information and certificates of the node itself.

1130 After received data has been validated (e.g. check that none of the received component ID
1131 already exist in another sub-network), the update in directory shall be a transaction.

1132 3.15.3 Updating the synchronization nodes' list

1133 The network administrator is responsible to build and send to all the node administrators the
1134 list and the access information of all the nodes of the network, namely for each node:

- 1135 • The node component ID.
- 1136 • The node access URLs (primary, secondary).

1137 The list is a single file, referred as the node-list file, whose format is described in § 5.4.

1138 A node administrator shall import the file to update the nodes to synchronize with.

- 1139 • The node shall process the file as a transaction, i.e. any error (e.g. incorrect format, non-
1140 unique component ID, missing certificate and internal error) shall cause the rollback of the
1141 whole update process, and the directory data shall remain unchanged.
- 1142 • The importation process shall ignore information about the current node which is included
1143 in the list.
- 1144 • The synchronization process which may update the current node directory shall be
1145 stopped during the importation.

1146 After importation, the node administrator can restart the synchronization process to update
1147 the directory with the reference data of the nodes.

1148 A node shall memorize the last time it successfully retrieved the data of each node of the
1149 node-list file. Such information shall be accessible by the administrator.

1150 NOTE A message exchanged between two endpoints having different home nodes can only be delivered correctly
1151 (including acknowledgements) after the two nodes have synchronized with each other at least once.

1152 3.16 Certificates and directory management

1153 3.16.1 Definitions and principles

1154 The security of a MADES network is based on a Public Key Infrastructure (PKI). Such
1155 infrastructure binds certificates both to the network components and to the parties using the
1156 network. Indeed the components cross-check their identities before exchanging information,
1157 the sender parties want that the only intended recipients can read the documents, and each
1158 party want to authenticate the senders of the documents he received.

1159 Certificates use asymmetric cryptography based on private and public keys. On the contrary
1160 of symmetric cryptography, encoding is done using one key and decoding using the other key
1161 (which is different, and hence the asymmetry). Where the public key can easily be deduced
1162 from the private key, the reverse operation is a very complex mathematical challenge. RSA
1163 algorithm is generally used for encoding and decoding.

1164 Encryption:

- 1165 • A document is encrypted⁵ when it is encoded with a randomly generated symmetric key.
1166 The key is attached to the document in a secret way, being encoded itself with the
1167 recipient's public key.
- 1168 • To decrypt the document, the recipient shall first decode the "encoded symmetric key"
1169 using its private key, and then decode the document with the symmetric key.

1170 Signing:

- 1171 • A signature is an encoded fingerprint of a list of resources. The list is referred as the
1172 signature manifest. The technical word for the fingerprint is a "hash", which is generated
1173 via a strong one-way transformation (e.g. SHA-1, SHA-512). The exact manifest of a
1174 MADES message is described in § 3.14.3.
- 1175 • The algorithm used to generate the hash does not require any key, so anyone having the
1176 manifest can generate the hash. Building another meaningful manifest generating the
1177 same hash is also a complex mathematical challenge. The signature is the hash encoded
1178 with the sender's private key.
- 1179 • Thus anyone having the manifest, the signature and the sender's public key can verify that
1180 the manifest is the one that was manipulated by the sender when he generated the
1181 signature. The sender cannot repudiate a manifest he signed.
- 1182 • Signing refers to the full process, i.e. generating the hash and encoding it.
- 1183 • Verifying a signature includes: regenerating the hash from the manifest, decoding the
1184 signature, and checking that both results are equal.

1185 A Certificate Authority (CA) is an entity that issues certificates.

1186 A certificate:

- 1187 • contains a public key, a name;
1188 • is signed with the private key of the certificate of the issuer CA;
1189 • has an expiration date, which is sooner than the expiration date of the certificate of the
1190 issuer CA.

1191 When signing a certificate, the issuer CA certifies the ownership of the keys (private and
1192 public) by the party whose name is in the certificate. Other parties can verify the certificate
1193 signature using the certificate of the issuer CA. So, if parties trust a CA, they can then rely
1194 upon the signatures generated using the certificates that the CA has issued.

1195 The certificate of a CA may itself have been issued and signed by another CA, the later
1196 delegating to the first the right to issue certificates. The certification chain of a certificate
1197 shows the delegation sequence of CAs: it is the list of the certificates of all CAs' from the
1198 issuer CA until an unsigned or self-signed certificate, referred as the root certificate.

1199 A valid certificate is a non-expired certificate. An expired certificate shall not be used for
1200 authentication, encryption or for signing a document. However, it can still be used to decrypt
1201 old documents or verify their signatures, and thus to prove whatever may be necessary.

⁵ Beware that "Encoding" and "Encryption" are not synonymous here. "Encoding" refers to an algorithmic operation, while "Encryption" is the process described here which ensures confidentiality. Both "Encryption" and "Signing" processes use "Encoding" operations.

1202 **3.16.2 Certificates: Format and unique ID**

1203 All components (endpoints and nodes) shall use certificates to be authenticated by their
1204 communication peers (transport-layer security), to sign and to encrypt the messages
1205 (message-level security) when necessary.

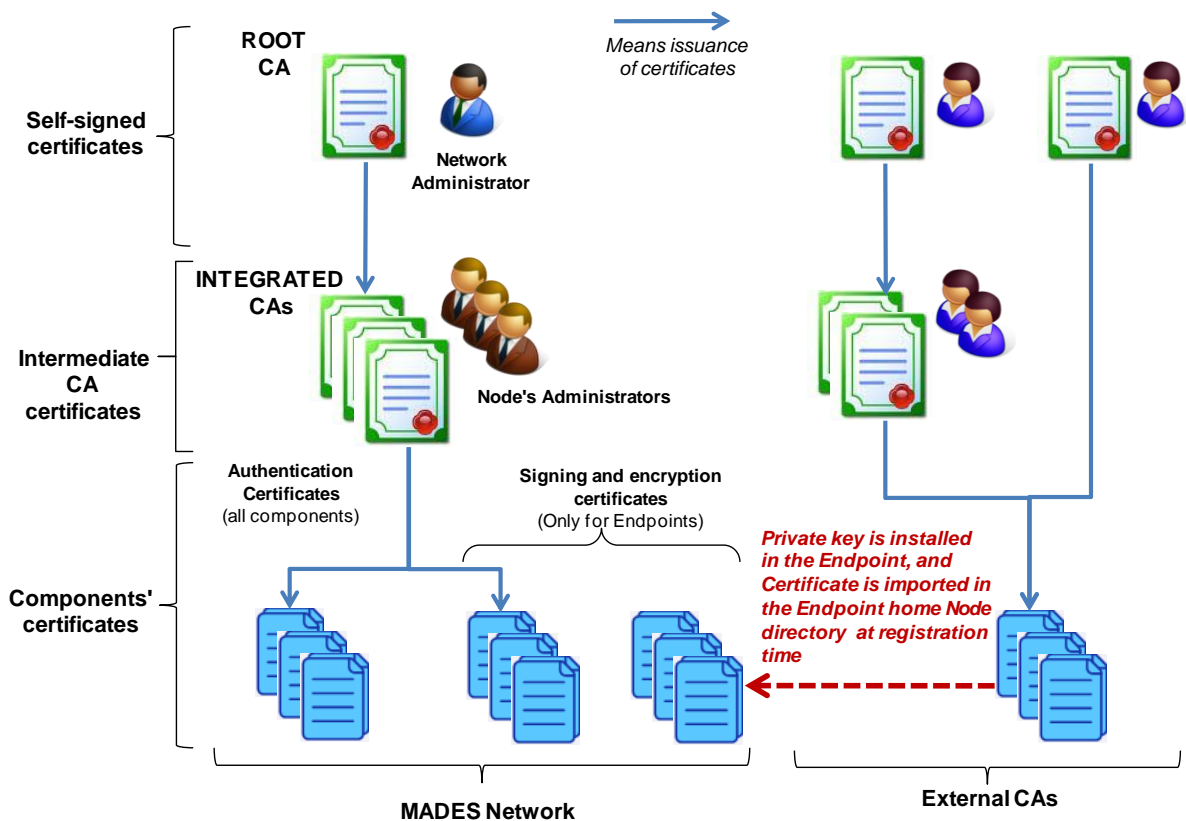
1206 The format of the certificates shall comply with the X.509 ITU-T standard, and the certificates'
1207 keys shall have a length of 2 048 bits.

1208 The exact concatenation of the standardized attributes “issuer” and “serial number” of a
1209 certificate forms a unique ID, referred as the “certificate ID”.

1210 **3.16.3 Used certificates and issuers (CAs)**

1211 **3.16.3.1 Overview**

1212 Figure 22 describes the certificate authorities and the certificates used in a MADES network.



1213

1214 **Figure 22 – Certificates and certificate authorities (CAs) for a MADES network**

1215 **3.16.3.2 Transport-layer security (Authorize data exchanges)**

1216 Each component (endpoint and node) shall own an authentication certificate published in the
1217 home node directory.

1218 The authentication certificates are issued by the network organization as follows:

- 1219
- The organization owns a ROOT CA certificate.

1220 • The organization delegates to each node administrator the right to issue the authentication
1221 certificates of the registered endpoints. Each administrator owns an INTEGRATED CA
1222 certificate issued by the ROOT CA.

1223 Each component shall store the authentication certificate and the corresponding private key.
1224 The authentication certificate is used whether the component acts as a client or a server.

1225 Whatever the operation using the authentication certificate, it shall fail when the certificate
1226 has expired.

1227 **3.16.3.3 Message-level security (Protect message confidentiality and authenticate**
1228 **message issuer)**

1229 Each endpoint shall own an encryption certificate published in the home node directory for the
1230 others endpoints to encrypt the business-messages they sent. The endpoint uses the
1231 corresponding private key to decrypt the business-messages it receives.

1232 Each endpoint shall own a signing certificate published in the home node directory for the
1233 others endpoints to verify the signature of the messages they receive. The endpoint uses the
1234 corresponding private key to sign the messages it sends.

1235 The signing and encryption certificates can be issued by the home node administrator using
1236 the INTEGRATED CA, or by an EXTERNAL CA trusted by the network parties and not
1237 necessary issued by one of the main public trusting organizations.

1238 The signing certificate of an endpoint can either be stored locally or inserted in a
1239 coding/decoding external device (e.g. smart cards).

1240 The endpoint shall never encrypt or sign a message using an expired certificate. The endpoint
1241 shall not decrypt or verify a message signature using a certificate that was expired at the time
1242 the message was created (creation time is included in the message header).

1243 Every endpoint shall durably store the signing certificates of the other endpoints in order to
1244 possess all necessary evidence.

1245 **3.16.4 Directory services**

1246 **3.16.4.1 Content and updates**

1247 Each node shall contain a directory where all the network components are described. Each
1248 entry for a component in the directory shall include:

- 1249 • the component ID (non-significant);
1250 • the component display name (human readable);
1251 • the component type (endpoint, node);
1252 • the technical contact information for operation or administration: name of the responsible
1253 person, e-mail and phones; the latter should be non-personal (hotline, operation centre,
1254 functional/generic e-mail);
1255 • the certificates owned by the component (one or several for each purpose including
1256 authentication, signing, and/or encryption – when applicable).

1257 The node administrator shall be able to update the directory entry of any of the registered
1258 components. This includes registering, updating and removing components, importing or
1259 renewing certificates for components. The description of the other nodes and their registered
1260 components is imported using the node synchronization — see § 3.15.2 and § 3.15.3.

1261 **3.16.4.2 Queries**

1262 The endpoints shall query their home node directory to get information on a component and to
1263 retrieve certificates (e.g. to encrypt a message, to verify a message signature or to
1264 authenticate a component that signed a token or a component ID) — see § 5.3.4.

1265 **3.16.5 Caching directory data**

1266 To reduce the request flow on the node directory, the endpoints shall implement a caching
1267 mechanism for directory data.

1268 A node shall implement a TTL (Time-To-Live) mechanism, whose duration is configurable. It
1269 shall provide an expiration time for any dataset returned by a directory request: the time when
1270 the request is processed + the TTL value.

1271 An endpoint shall not use the expired data in cache and shall then request again the home
1272 node for the data.

1273 **3.16.6 Trusting the certificates of others components**

1274 **3.16.6.1 Authentication**

1275 A component shall only communicate with a peer component if the authentication certificate
1276 presented by the peer component:

- 1277 • belongs to the ROOT CA certification chain;
1278 • is successfully verified.

1279 During the TLS authentication phase, each peer shall convey to the other the following
1280 ordered certificate chain:

- 1281 1. Its own authentication certificate.
1282 2. The INTEGRATED CA certificate certified by the ROOT CA and which certifies the
1283 authentication certificate.

1284 A component shall trust the ROOT CA and any authentication certificate provided by the
1285 home node (e.g. used for token-authentication).

1286 **3.16.6.2 Signing and encryption**

1287 The endpoint shall trust the signing and encryption certificates provided by the home node.

1288 **3.16.7 Renewing the expired certificates**

1289 **3.16.7.1 Renewing the authentication certificates**

1290 In case the authentication certificate of a component is renewed, the component will convey
1291 the certificate, possibly certified by a new INTEGRATED CA, to the peer component during
1292 the TLS authentication phase. When the INTEGRATED CA certificate is signed by the
1293 ROOT CA certificate, the communication is possible.

1294 Additionally every component shall be configurable to communicate with components whose
1295 certificates may belong to two distinct certificate chains. So when the ROOT CA is renewed,
1296 components can communicate whether their certificate belongs to the old or to the new
1297 certification chain.

1298 **3.16.7.2 Renewing process (authentication, signing and encryption):**

- 1299 • The issuer CA shall renew the certificate enough time before the old one expires so that
1300 their validity periods overlap.
- 1301 • The new certificate is published (imported) into the component home node.
- 1302 • The new certificate is then installed into the component (including the private key), before
1303 the old certificate expires.

1304 To do so:

- 1305 • A node directory shall be able to store two certificates of any type for a component.
- 1306 • Until the old encryption certificate expires, a node shall not provide the new one when
1307 replying to a directory request, for it may not have been installed into the owner
1308 component — see rules for the *GetCertificate* service; § 5.3.4.2.
- 1309 • An endpoint shall be able to contain simultaneously two encryption certificates (private
1310 key).

1311 Installing a new certificate into a component shall not last more than 5 minutes to ensure
1312 business continuity.

1313 **3.16.8 Revoking a certificate**

1314 A certificate shall only be revoked for security reasons when there is a reasonable doubt that
1315 it could be misused.

1316 Revoking a certificate is a request to the certificate issuer. As a result, the issuer usually
1317 inserts the certificate serial number in a Certificate Revocation List (CRL), which can be
1318 publicly accessed. MADES does not implement such a CRL mechanism.

1319 Within a MADES network, revoking a certificate is a request to the administrator of the node
1320 where the endpoint has registered. The node administration tool shall provide the ability to
1321 revoke any certificate of a registered endpoint. The certificate shall then be tagged as
1322 revoked in the node directory⁶. This tag is:

- 1323 • propagated to others nodes by the node synchronization mechanism;
- 1324 • used to decide whether a requested certificate is delivered or not — see § 5.3.4.2.

1325 The node administrator shall be able to revoke a certificate either issued by the
1326 INTEGRATED CA or by an EXTERNAL CA.

1327 Such revocation of a certificate issued by an EXTERNAL CA has no link with the revocation
1328 process stated by the issuer in his certificate policy. The certificate owner shall also and
1329 always and independently ask for the certificate revocation by the certificate issuer. No
1330 MADES components ever access to any CRL of an external issuer.

1331 The consequences of a certificate revocation, resulting from the message delivery mechanism
1332 described in the previous sections, are summarized in Table 10:

⁶ This shall increase the directory Dversion — see § 3.15.2.

1333

Table 10 – Consequences of a certificate revocation

| Revoked certificate | Consequences |
|-------------------------------------|---|
| Endpoint signing certificate | <p>From the revocation moment, all business-messages and all signed acknowledgements coming from the endpoint and uploaded to the home node will be <u>rejected</u>.</p> <p>The business-messages and the signed acknowledgements coming from the endpoint and uploaded to another node will be <u>rejected</u> after the node has synchronized with the endpoint home node.</p> <p>The messages to and from the endpoint pending in a node (home or not) before the revocation tag is updated in the node, will be <u>delivered</u>.</p> |
| Endpoint encryption certificate | <p>From the revocation moment, all business-messages for the endpoint and uploaded to the home node will be <u>rejected</u>.</p> <p>The business-messages for the endpoint pending in the home node before the revocation moment will be <u>delivered</u> to the endpoint.</p> |
| Endpoint authentication certificate | <p>From the revocation moment, the home node will <u>reject downloading</u> messages for the endpoint. Those messages will be delivered (if not expired) after the endpoint has renewed the certificate.</p> |

1334 NOTE The process to renew a revoked certificate is defined by the network governance.

1335 4 Managing the version of the MADES specification

1336 4.1 Issues and principles

1337 4.1.1 General

1338 When the MADES specification changes from version N-1 to version N, a MADES network
1339 should then upgrade to the new version.

1340 A “big bang” rollout on all components would be both complex to coordinate and risky
1341 regarding business continuity, and thus unacceptable.

1342 A smooth rollout means that an upgraded endpoint can successfully exchange messages with
1343 a non-upgraded endpoint (using version N-1), and that two upgraded endpoints can
1344 successfully exchange messages using new version N.

1345 This clause shows how such a rollout shall be done and the constraints that any new version
1346 of the specification should satisfy.

1347 4.1.2 Rolling out a new version (Mversion and N-compliance)

1348 The rollout of a new version shall start with nodes. An endpoint shall only upgrade after the
1349 home node did.

1350 A node upgraded to version N can successfully process the requests from the non-upgraded
1351 endpoints if and only if it still exposes interfaces compliant with version N-1. So, as a general
1352 rule, a node upgraded to version N shall still expose the N-1 compliant interfaces. Also the
1353 upgraded nodes shall still request, being clients, the non-upgraded nodes for synchronization.

1354 A component is referred as N-compliant when it complies with version N of the MADES
1355 specification, and when it can successfully transfer messages with components which comply
1356 with version N-1.

1357 From version 2, every component shall be N-compliant. This means that it complies with a
1358 version and can exchange using the previous version.

1359 Every component shall access the installed version to which it complies, referred as the
1360 Mversion (MADES version).

1361 Notations:

1362 • A N-service or a N-interface is a service or an interface that complies with the version N of
1363 the MADES specification.

1364 • A N-component is a N-compliant component (e.g. N-node, N-endpoint). Note that a N-
1365 component exposes (as server) or uses (as client) N-services and N-1-services.

1366 • A N-message is a message composed according to the version N of the MADES
1367 specification.

1368 4.1.3 Service compatibility

1369 A N-component server exposes both N-interface and N-1-interface. This does not mean that
1370 the N-interface is completely new (e.g. some services may not change).

1371 It is up to the specification team to decide which and how the functions, the interfaces and the
1372 services evolve. The possible changes for a service are listed in Table 11:

1373 **Table 11 – Service compatibility – Possible changes**

| N° | Service changes |
|----|--|
| 1 | The service does not change. |
| 2 | The description of the service does not change but the way the elements are used in queries and responses does change, e.g.: <ul style="list-style-type: none"> • Some previously technically optional elements are now functionally required. • New values are now possible in the elements (e.g. new encryption algorithm using different metadata). |
| 3 | The description of the service changes in a compatible way, e.g.: <ul style="list-style-type: none"> • A new optional element is created. • A mandatory element becomes optional. • An unused optional element is removed. |
| 4 | The description of the service changes in a non compatible way → Actually, it is a new service with a new name. |

1374 In order to allow the specification team to use all these possibilities, most services include a
1375 “*serviceMversion*” element as part of the request. So the service behaviour can change
1376 without creating a new service, provided the client uses that element to tell the server which
1377 version of the specification it is working with. The server can then process the request and
1378 reply as expected.

1379 4.1.4 Message compatibility

1380 The description of a message or the way a message is composed may evolve from version N-
1381 1 to version N. When this happens, a N-1-component will probably fail to process a N-
1382 message.

1383 To ensure that a sender’s endpoint composes a message that a recipient’s endpoint can
1384 understand, the principles are the followings:

1385 • The node directory shall store (dynamically) the installed Mversion of the registered
1386 endpoints, which is then transferred to other nodes through the synchronization process.

- 1387 • Each endpoint shall notify its installed Mversion to the home node when starting using the
1388 *SetComponentMversion* service.
- 1389 • The directory services shall provide the installed Mversion of an endpoint.
- 1390 • The description of a message contains a *messageMversion* element which tells the
1391 version of the specification to which the message complies.
- 1392 • The transfer (upload and download) N-services shall mix N-messages and N-1-messages,
1393 e.g. the collection of messages transferred in the *UploadMessage* request can contain
1394 both N-messages and N-1 messages.
- 1395 • A sender's endpoint shall compose a business-message that the recipient's endpoint can
1396 understand — see detailed rules in § 4.2.4.
- 1397 • A component shall compose an acknowledgment using the same Mversion as the original
1398 message.
- 1399 • In case a component receives a message that it cannot process:
- 1400 a) It shall reject the message, while confirming the transfer when a node.
- 1401 b) Otherwise it shall log the error and (if possible) it may store the message in the
1402 FAILED state, and shall issue a failure-event.

1403 4.1.5 Interface with BAs

1404 The BAs are not concerned with the MADES specification version; so the used Mversion is
1405 not an element of the endpoint interface.

1406 A change in a service of the endpoint interface should be backward compatible; otherwise the
1407 new specification should create a new service, and both (old and new) services should be
1408 described in the new specification. New BAs would then use the new service, and existing
1409 applications would migrate to the new service. Thus the migration timescale for the BAs can
1410 be kept independent of the network components' upgrade.

1411 An endpoint administrator shall be able to configure an association between a business-type
1412 and a minimum required Mversion. The default value is 1. It can be used when some new
1413 features available from this version are required for the business process (e.g. new encryption
1414 algorithm).

1415 4.2 Using the correct version for services and messages

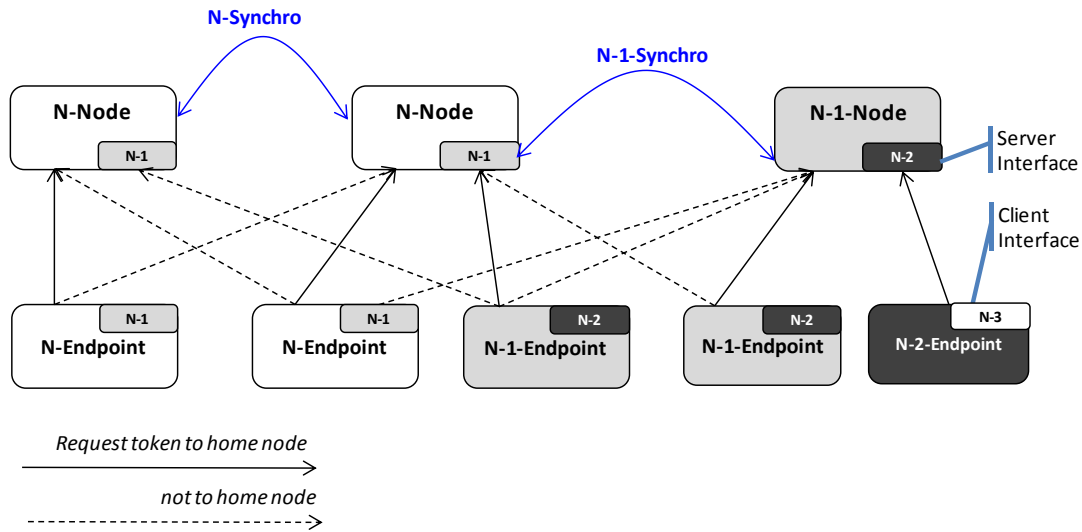
1416 4.2.1 Node synchronization and authentication

1417 Figure 23 shows which version of the authentication and synchronisation service is used
1418 between components. A N-component server also exposes a N-1-interface and, acting as a
1419 client can request the N-1-interface of a N-1-component server.

1420 Node synchronization:

- 1421 • A node shall request and store the Mversion of each node of the node-list file.
- 1422 • The *GetNodeMversion* service (see § 5.3.5.1) shall be used by a node to get the Mversion
1423 of a node of the node-list file, each time the node (re)starts and each time the node-list file
1424 is updated.
- 1425 • A N_A -node shall stop synchronizing with a N_B -node when $|N_A - N_B| > 1^7$.
- 1426 • When requesting for a N_B -node directory data using the *GetAllDirectoryData* service (see
1427 § 5.3.5.2), a N_A -node shall use the N-service, where $N = \text{Min}(N_A, N_B)^8$.

⁷ |a|: means "absolute value" (or "modulus") of a.



1428

1429

1430

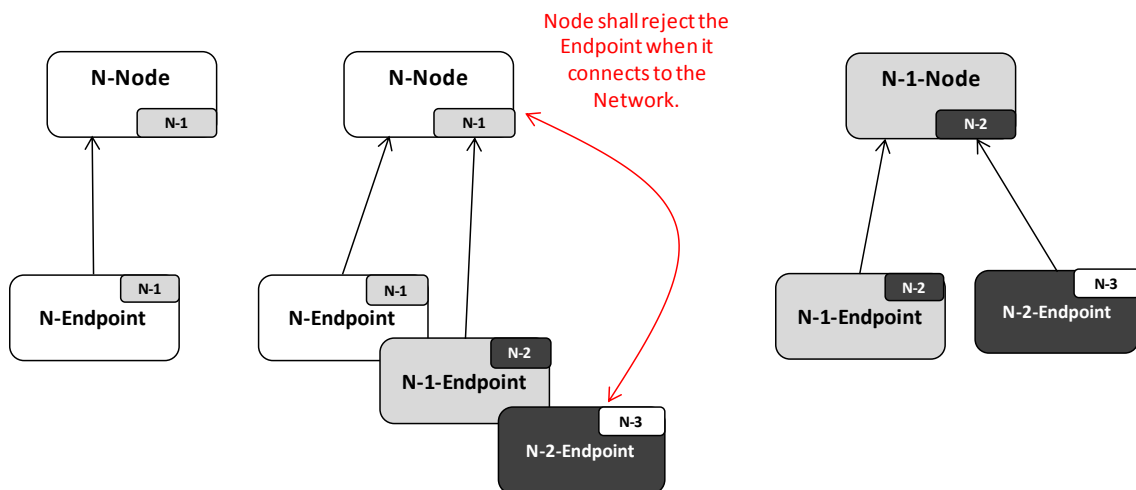
Figure 23 – Managing the specification version – node synchronization and authentication

1431 Requesting for an authentication-token:

- 1432 • A N_E -endpoint shall request for a token to the home node using the N_E -service.
- 1433 • A N_E -endpoint shall request for a token to another N_N -node using the N -service where
- 1434 $N = \text{Min} (N_E, N_N)$. The Mversion of the node is available in the home directory with the
- 1435 node routing information.
- 1436 • A N_N -node shall reject the authentication request from a N_E -endpoint when $(N_E > N_N)$ or
- 1437 $(N_E < N_N - 1)$.

1438 **4.2.2 Directory services and Network acceptance**

1439 Figure 24 describes the management of different MADES version in a MADES network.



1440

1441

Figure 24 – Managing the specification version – Directory services

1442 A N_E -endpoint shall always use the N_E -interface when requesting a directory service.

⁸ Min (a, b): means "minimum" value of a and b.

1443 After an endpoint has obtained an authentication-token from the home node, it shall always
1444 request for acceptance in the network.

1445 To do so, the component uses the *SetComponentMversion* service (see § 5.3.4.1) to inform
1446 the server about its installed Mversion. The reply informs the endpoint whether it is accepted
1447 or rejected by the network.

1448 A rejected component shall log the error and stop running.

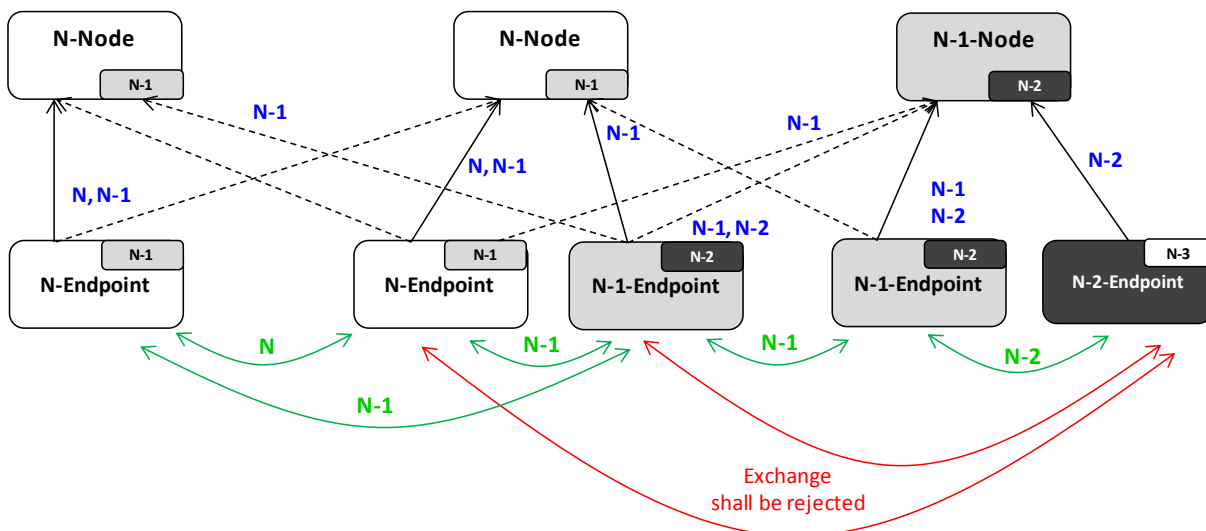
1449 Acceptance by the home node

- 1450 • A N_N -node shall reject a N_E -endpoint when either:
 - 1451 a) ($N_E > N_N$) or ($N_E < N_N - 1$);
 - 1452 b) the node cannot authenticate the endpoint (i.e. incorrect signed endpoint ID);
 - 1453 c) the endpoint did not register with the node;
- 1454 • Otherwise the node shall accept the endpoint, store the endpoint Mversion in the
1455 directory, increase the directory data version (Dversion) if N_E stored value changes, and
1456 reply providing its own Mversion (N_N).
- 1457 • A node shall log that the Mversion of an endpoint has changed; or that the endpoint has
1458 been rejected.

1459 When a node restarts, after the installed version has changed, no session information (e.g.
1460 token) from the previously connected endpoints shall remain. This ensures that all endpoints
1461 will newly request for network acceptance.

1462 NOTE In case an N-1-endpoint is stopped, other endpoints will continue to send it N-1-messages. When it comes
1463 back to the network, being upgraded to version N, other endpoints will still continue to send it N-1-messages until
1464 their directory cache (see § 3.16.5) is renewed. But the endpoint will process correctly those N-1-messages. Only
1465 the pending N-2-messages will be rejected, but anyway the endpoint cannot exchange anymore with those N-2
1466 peers until they upgrade.

1467 **4.2.3 Messaging services**



1468

1469 **Figure 25 – Managing the specification version – Messaging services**

1470 The Figure 25 shows:

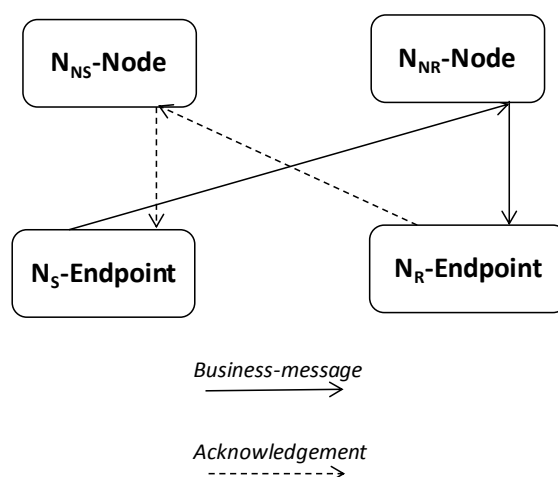
- 1471 • the messaging services that shall be used between components and the possible
1472 Mversion of the transferred messages (in blue);

1473 • the endpoints that can exchange messages and the required Mversion for the exchange
1474 (in green).

1475 Figure 25 presents a situation where two endpoints cannot exchange although they only have
1476 1 version difference (N-2, N-1). The reason is that the N-1-endpoint has registered with a N-
1477 node. And the N-node will reject any N-2-message either a business-message or an
1478 acknowledgement.

1479 4.2.4 Which version to use to send a message?

1480 Figure 26 describes the way of handling different versions of MADES and Table 12 provides
1481 the meaning of the references.



1482

1483 **Figure 26 – Managing the specification version –**
1484 **Which version to use to send a message?**

1485 **Table 12 – Which version to use to send a message?**

| Mversion | |
|---------------------|--|
| N_S | The Mversion of the sender's endpoint. |
| N_{NS} | The Mversion of the home node of the sender's endpoint. |
| N_R | The Mversion of the recipient's endpoint. |
| N_{NR} | The Mversion of the home node of the recipient's endpoint. |
| N_B — see § 4.1.5 | The minimum Mversion required for the business-type |

1486 The used version should be $N = \text{Min}(N_S, N_R)$, however the message shall be rejected if one
1487 of the conditions listed in Table 13 is verified:

1488 **Table 13 – Managing the specification version – Rejection conditions**

| Condition | Reason for rejection |
|-------------------|--|
| N_R unknown | The MADES version of the recipient's endpoint is unknown. |
| $ N_R - N_S > 1$ | The sender's endpoint and the recipient's endpoint are not MADES compatible. |

| Condition | Reason for rejection |
|----------------------|---|
| $ N_{NR} - N_S > 1$ | The sender's endpoint is not MADES compatible with the recipient's node |
| $ N_{NS} - N_R > 1$ | The recipient's endpoint is not MADES compatible with the sender's node. |
| $N_B > N_S$ | The sender's endpoint is not MADES compatible with the minimal version required for the business-type |
| $N_B > N_R$ | The recipient's endpoint is not MADES compatible with the minimal version required for the business-type. |

1489 **5 Interfaces and services**

1490 **5.1 Introduction**

1491 **5.1.1 General**

1492 This chapter describes all services for components to exchange each other or with the BAs.
1493 The description provides all elements in a request and the corresponding reply independently
1494 of an implementation language.

1495 **5.1.2 Error Codes**

1496 In case a service encounters an unrecoverable error, it returns information on the error. When
1497 not described the set of the returned elements is listed in Table 14 and the errorCode values
1498 are listed in Table 15

1499 **Table 14 – Interfaces and services – Generic error**

| Element name | Description | Element type |
|--------------|---|--------------|
| errorCode | A code representing the type of error. | string |
| errorID | Unique identification of the error. | string |
| errorMessage | An English readable text describing the error. | string |
| errorDetails | (optional) Additional English readable details about the error context. | string |

1500 **Table 15 – Interfaces and services – String value for errorCode**

| String value for errorCode | Description |
|----------------------------|---|
| INVALID_PARAMETERS | The provided parameters (i.e. request elements) are incomplete, are not in the expected format or do not have the expected syntax. |
| AUTHENTICATION_ERROR | The peer component cannot be authenticated |
| VALIDATION_ERROR | The message is not valid (content size exceeded, unknown sender/recipient, signature is not valid etc.). |
| INTERNAL_ERROR | Internal application error. The error was not caused by the content of request but by the application itself (<i>Null Pointer Exception</i> in code, full database etc.) |
| CONCURRENT_ERROR | The server component is already processing a concurrent request from the same client. |

1501 **5.1.3 Types for Time**

1502 All date and time shall be expressed in UTC (Coordinated Universal Time). The used time
1503 types are:

- 1504 • “timestamp” technically means “xsd:long”, and the value is the number of milliseconds
1505 since 'midnight 1.1.1970 UTC'.
- 1506 • “dateTime” technically means “xsd:dateTime” and the value is according to the XSD
1507 specification (<http://www.w3.org/TR/xmlschema-2/#dateTime>).

1508 **5.2 Endpoint interface**

1509 **5.2.1 Overview**

1510 The endpoint interface provides the business applications (BAs) with the access to the
1511 MADES communication network.

1512 MADES specifies this interface using Web services – The BA calls the web services exposed
1513 by the endpoint.

1514 There are 5 available services:

- 1515 • *SendMessage* — used to upload a message into the endpoint in order to send it to another
1516 endpoint.
- 1517 • *ReceiveMessage* — used to download a message from the endpoint.
- 1518 • *CheckMessageStatus* — used to check the current delivery status of a message.
- 1519 • *ConnectivityTest* — used to check if another endpoint can be reached.
- 1520 • *ConfirmReceiveMessage* — used to notify the endpoint that a received message has been
1521 technically accepted by a BA.

1522 The BAs can access the network using files. This interface is called FSSF (File System
1523 Shared Folders) and is described in § 5.2.3.

1524 **5.2.2 Services**

1525 **5.2.2.1 SendMessage service**

1526 The *SendMessage* service is used by a BA to upload a message into the endpoint in order to
1527 send it to another endpoint.

1528 The service request elements are provided in Table 16.

1529 **Table 16 – SendMessage – Service request elements**

| Element name | Description | Element type | Required |
|----------------|--|----------------------------|----------|
| message | Sending context, content and requested destination of the message. | SendMessage (see Table 76) | True |
| conversationID | Unique identifier associated with the request. | string | False |

1530 Concerning *conversationID*: There are situations where the sender’s BA may not receive back
1531 or may fail to durably store the returned message ID, for example in case of failure of the
1532 endpoint, of the network or of the BA itself. So the BA does not know if the message was or
1533 was not correctly transferred to the sender’s endpoint. There are two subsequent issues:

- 1534 • If the message was actually correctly transferred and stored in the endpoint, the BA does
1535 not know the message ID needed for further processing, such as checking the delivery
1536 status of the message.

- 1537 • Considering that losing a message is a non acceptable risk, the BA will send the message
1538 again when the connection with the endpoint is restored. The drawback is that the same
1539 message may then be sent twice with two different IDs.

1540 So, resending the message using the same *conversationID* value solves both issues. Indeed,
1541 when an endpoint is requested to send a message with a *conversationID* value that has
1542 already been used for an existing stored and sent message, it shall not send the message
1543 again but return the caller BA with the ID of the already existing message. The
1544 recommendation is that *conversationID:= senderApplication + baMessageID*.

1545 The service response elements are provided in Table 17.

1546 **Table 17 – SendMessage – Service response elements**

| Element name | Description | Element type |
|--------------|--|--------------|
| messageID | The UUID (Universal Unique ID) of the message composed and stored by the endpoint — see § 3.2. | string |

1547 Additional⁹ error elements for the service are listed in Table 18.

1548 **Table 18 – SendMessage – Additional error elements**

| Element name | Description | Element type |
|--------------|---|--------------|
| receiverCode | The component ID of the requested recipient's endpoint for the message. | string |

1549 5.2.2.2 ReceiveMessage Service

1550 The *ReceiveMessage* service is used by a BA to download a message from the endpoint.

1551 The service request elements are provided in Table 19.

1552 **Table 19 – ReceiveMessage – Service request elements**

| Element name | Description | Element type | Required |
|-----------------|--|--------------|----------|
| businessType | The business-type of the requested message — see § 3.3. Pattern: [A-Za-z0-9]+ ¹⁰ | string | True |
| downloadMessage | The service returns, if any, the first received and pending message having the requested business-type. "First" means according to the priority defined in § 3.13. The content (or document) of the message is or is not returned according to the value of the element: true:= returned; false:= not returned. | boolean | True |

1553 The service response elements are provided in Table 20.

⁹ In addition to the elements described in § 5.1.2.

¹⁰ Pattern is the « regular expression » that the element value shall match.

1554 **Table 20 – ReceiveMessage – Service response elements**

| Element name | Description | Element type |
|------------------------|--|--------------------------------|
| receivedMessage | Sending context and possibly content of a message. | ReceivedMessage (see Table 74) |
| remainingMessagesCount | The number of remaining messages received by the endpoint, matching the requested business-type and waiting for delivery. In case the service returns the content of a message, the message is not included in the count of the remaining messages. | integer |

1555 Additional error elements for the service are provided in Table 21.

1556 **Table 21 – ReceiveMessage – Additional error elements**

| Element name | Description | Element type |
|--------------|---------------------------------------|--------------|
| businessType | The business-type that was requested. | string |

1557 Until the recipient's BA confirms to the recipient's endpoint that the message is correctly
1558 transferred using the *ConfirmReceiveMessage* service, the endpoint shall consider that the
1559 message has not been transferred, but is still pending and shall be transferred again next
1560 time a BA requests for the business-type. This ensures that no message may be lost. As a
1561 consequence the BAs shall be aware that, in some failure or recovery situations, they may
1562 possibly receive an already delivered message (i.e. having a known message ID).

1563 5.2.2.3 CheckMessageStatus Service

1564 The *CheckMessageStatus* service is used to check the current delivery status of a message.

1565 The service request elements are provided in Table 22.

1566 **Table 22 – CheckMessageStatus – Service request elements**

| Element name | Description | Element type | Required |
|--------------|--|--------------|----------|
| messageID | The UUID (Universal Unique ID) of the message whose status is requested — see § 3.2. | string | True |

1567 The service response elements are provided in Table 23.

1568 **Table 23 – CheckMessageStatus – Service response elements**

| Element name | Description | Element type |
|---------------|---|---------------------------|
| messageStatus | All Information about the message delivery. | MessageStatus (see § 5.5) |

1569 Additional error elements for the service are provided in Table 24.

1570 **Table 24 – CheckMessageStatus – Additional error elements**

| Element name | Description | Element type |
|--------------|---------------------------|--------------|
| messageID | The requested message ID. | string |

1571 **5.2.2.4 ConnectivityTest Service**

1572 The *ConnectivityTest* service can be used to check if another endpoint can be reached. The
1573 service just sends a tracing message whose delivery-status can further be requested using
1574 the *CheckMessageStatus* service. The connectivity is successful, i.e. the tracing-message
1575 has reached the recipient's endpoint, when the status is DELIVERED.

1576 The service request elements are provided in Table 25.

1577 **Table 25 – ConnectivityTest – Service request elements**

| Element name | Description | Element type | Required |
|--------------|--|--------------|----------|
| receiverCode | The component ID of the recipient's endpoint whose connectivity is checked. Pattern: [A-Za-z0-9-@]+ | string | True |

1578 The service response elements are provided in Table 26.

1579 **Table 26 – ConnectivityTest – Service response elements**

| Element name | Description | Element type |
|--------------|--|--------------|
| messageID | The message ID of the tracing-message. | string |

1580 Additional error elements for the service are provided in Table 27.

1581 **Table 27 – ConnectivityTest – Additional error elements**

| Element name | Description | Element type |
|--------------|--|--------------|
| receiverCode | The component ID of the recipient's endpoint whose connectivity check was requested. | string |

1582 **5.2.2.5 ConfirmReceiveMessage service**

1583 The *ConfirmReceiveMessage* service is used by a recipient's BA to confirm the download
1584 transfer of a message from the recipient's endpoint.

1585 A BA cannot reject a message; the business functional acceptance (i.e. compliance with
1586 business rules) is another issue. If a message is not confirmed back, for example in case of
1587 failure, the endpoint will provide it again at the next *ReceiveMessage* call.

1588 In case a BA encounters an unrecoverable error when processing a transferred message, and
1589 when the error comes from the message itself (e.g. inconsistent elements) and not from the
1590 application (e.g. file system full), the BA should confirm the transfer, log the error and
1591 possibly alert, otherwise the message will indefinitely be retransferred by the endpoint until it
1592 is confirmed.

1593 The service request elements are provided in Table 28.

1594 **Table 28 – ConfirmReceiveMessage – Service request elements**

| Element name | Description | Element type | Required |
|--------------|--|--------------|----------|
| messageID | The UUID (Universal Unique ID) of the message whose transfer to the BA is being confirmed — see § 3.2. | string | True |

1595 The service response elements are provided in Table 29.

1596 **Table 29 – ConfirmReceiveMessage – Service response elements**

| Element name | Description | Element type |
|--------------|--|--------------|
| messageID | The UUID (Universal Unique ID) of the message whose transfer has be confirmed. | string |

1597 Additional error elements for the service are provided in Table 30.

1598 **Table 30 – ConfirmReceiveMessage – Additional error elements**

| Element name | Description | Element type |
|--------------|---------------------------|--------------|
| messageID | The requested message ID. | string |

1599 **5.2.3 File System Shared Folders (FSSF)**

1600 **5.2.3.1 Introduction**

1601 The FSSF interface is the way for a BA to exchange documents as files with the endpoint.
1602 The file system where the files are written is accessed by the endpoint as local file system.
1603 The principles are the followings:

- 1604 • All the sender’s BAs write in a common and unique OUT-folder the documents that the
1605 endpoint shall send.
- 1606 • The recipient’s BAs read in an IN-folder the documents that the endpoint has received.
- 1607 • Additional information that is necessary for the message delivery is included in the
1608 filenames. Such information is the request/reply elements of the *SendMessage* and
1609 *ReceiveMessage* services.
- 1610 • The organisation of the directories is local to each endpoint and is configurable.

1611 When implemented, a file interface with the endpoint shall comply with the FSSF interface as
1612 described in the current section.

1613 NOTE There are differences between interfacing the endpoint using FSSF and using the webservice interface.

- 1614 • *CheckMessageStatus* and *ConnectivityTest* services are not supported.
- 1615 • *ConfirmReceiveMessage* is implicit; i.e. a message is moved to the RECEIVED state in
1616 the recipient’s endpoint when the content has been successfully written into a file in the
1617 IN-folder.
- 1618 • Actually FSSF, i.e. the processing of sending and receiving documents using files, may be
1619 considered —and also probably built— as a business application (BA) embedded with the
1620 endpoint.

1621 **5.2.3.2 Used files and file naming convention**

1622 There are 4 types of files used by the FSSF interface. Each file type is written into a separate
1623 folder.

1624 The filenames are built from several parts joined by underscores (“_”).

- 1625 • Each part is limited to alphanumeric or hyphen characters. Accented characters, white
1626 spaces, and special characters shall not be used.
- 1627 • Joining underscores shall be present even when optional part is missing or empty.

1628 Table 31 provides respectively the description and filename format and Table 32 the filename
1629 description.

1630

Table 31 – FSSF – Description and filename format

| Type | Folder / Writer | Description and Filename format |
|------------------|-------------------------------|--|
| Files to be sent | OUT / BAs | <p>The folder contains the files written by BAs to be sent by the endpoint.</p> <p>The sender's endpoint removes from the folder the files that it has processed correctly (i.e. accepted files are deleted) or not (i.e. rejected files are moved to the OUT_ERROR folder).</p> <p>Filenames: <SenderBA>_<Recipient>_<BusType>_<BAmessageID>.<Ext></p> |
| Failed files | OUT_ERROR / Sender's endpoint | <p>The folder contains the files that the sender's endpoint did not process correctly. They have been moved from the OUT-folder to the OUT-ERROR-folder without changing their names.</p> <p>It's up to the endpoint administrator to analyse and clean up the folder.</p> <p>The filenames can match or not the "files to send" filename format. Note that not matching the filename format is a reason for the file not to be processed correctly.</p> |
| Received files | IN / Recipient's endpoint | <p>Each file in the folder contains the content of a message that has been received by the recipient's endpoint. The filename is built from the header information of the received message.</p> <p>The files should be removed from the folder when processed, correctly or not, by the recipient's BAs.</p> <p>Filenames: <SenderBA>_<Sender>_<BusType>_<BAmessageID>_<MessageID>.<Ext></p> |
| Log files | OUT_LOG / Sender's endpoint | <p>The folder contains one log file for each message accepted by the endpoint.</p> <p>The file contains English readable text. Each line reports an event about the message delivery, and is the concatenation of the <i>MessageTraceItem</i> structure, as provided in the <i>CheckMessageStatus</i> service response.</p> <p>The file is appended with a new line each time a new event for the message is notified to the sender's endpoint.</p> <p>It's up to the endpoint administrator to clean up the OUT_LOG folder.</p> <p>The filename is the exact name of the sent file with an added ".log" extension: <SenderBA>_<Recipient>_<BusType>_<BAmessageID>.<Ext>.log</p> |

1631

Table 32 – FSSF – Filename description

| Filename parts | Type | Description |
|----------------|-----------|---|
| <BAmessageID> | Optional | An identifier of the document provided by the sender's BA. Information is transported "as is" to the recipient's BA. — Pattern: [A-Za-z0-9-]* |
| <BusType> | Mandatory | The business type for the message (see § 3.3). — Pattern: [A-Za-z0-9-]* |
| <Ext> | Optional | The file extension — Pattern: [A-Za-z0-9-]* |
| <MessageID> | Mandatory | The UUID (Universal Unique ID) of the message composed by the sender's endpoint (see § 3.2) — Pattern: [A-Za-z0-9-]+ |
| <Sender> | Mandatory | The component code of the sender's endpoint. — Pattern: [A-Za-z0-9-@]+ |
| <SenderBA> | Optional | The identifier of the sender's BA. — Pattern: [A-Za-z0-9-]* |
| <Receiver> | Mandatory | The component code of the recipient's endpoint. — Pattern: [A-Za-z0-9-@]+ |

1632 Additional rules:

- 1633 • The to-be-sent filenames without extension shall not end with the dot character (".").
- 1634 • The sender's endpoint shall ignore files in the OUT-folder with extension "TMP" (or "tmp").

1635 • The sender's endpoint shall fail to send the files that matches one of the following
1636 conditions:

1637 1) Filename does not match the "files to-be-sent" Filename format.

1638 2) Filename is longer than 200 characters.

1639 3) File is empty.

1640 **5.2.3.3 Concurrent access to files**

1641 **5.2.3.3.1 General**

1642 As the BAs and the endpoint concurrently access to the files, special attention is required to
1643 avoid access conflicts and data losses.

1644 **5.2.3.3.2 Access conflicts**

1645 To avoid access conflicts between the writer and a reader of the file:

1646 • The file-reader shall ignore files whose extension is "TMP" (or "tmp").

1647 • The file-writer shall write the data first in a temporary file whose extension is "TMP" (or
1648 "tmp"), and then rename it changing the extension (note: "rename" is an atomic operation
1649 on every file system).

1650 **5.2.3.3.3 Data losses**

1651 Data may be lost if a file is overridden by another file having the same filename. To avoid this,
1652 each file should have a unique filename:

1653 • OUT – OUT_ERROR – OUT_LOG: It is highly recommended that the BAs uses
1654 <SenderBA> and <BAmessageID> to ensure they cannot use the same filename.

1655 • IN: the use of <MessageID> in the filename ensures that the content of two different
1656 messages will always be written in 2 different files.

1657 **5.2.3.4 Configuring FSSF**

1658 The administrator shall be able to configure the endpoint with following information:

1659 • OUT folder name;

1660 • OUT_ERROR folder name;

1661 • OUT_LOG folder name;

1662 • A list of business-types, and for each one an associated folder name and a default
1663 extension.

1664 The recipient's endpoint shall write in files the content of the business-messages whose
1665 business-type is in the configured list:

1666 • The file shall be written in the IN folder which is associated with the message's business-
1667 type.

1668 • The file shall have the extension provided in the "extension" attribute of the message
1669 header — see Table 61. When none, the extension shall be the default extension for the
1670 message's business-type.

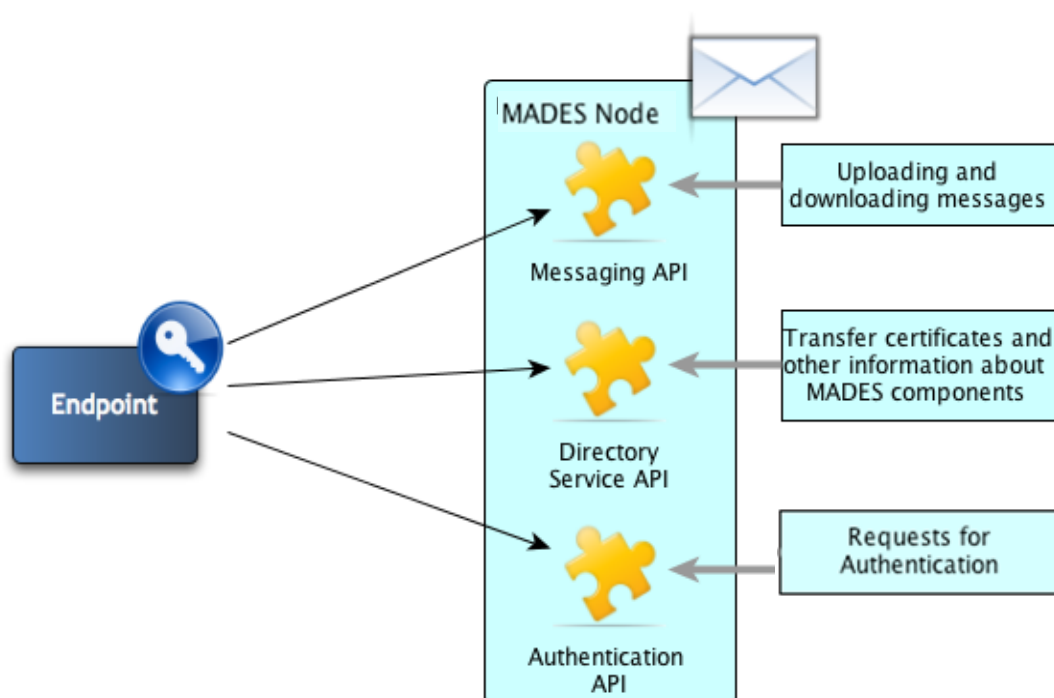
1671 **5.3 Node interface**

1672 **5.3.1 Overview**

1673 The node interface (see Figure 27) provides the endpoint access to the node. MADES
1674 specifies the interface exposed by the node using Web services – over SOAP/HTTPS
1675 protocol. The services are classified as follows:

- 1676 • Authentication service — see § 5.3.2.
- 1677 • Messaging services — see § 5.3.3.
- 1678 • Directory services — see § 5.3.4.

1679 The node synchronization interface is used by the nodes to synchronize their directory data
1680 each other. MADES specifies the interface using Web services – over SOAP/HTTPS protocol
1681 — see § 5.3.5.



1682

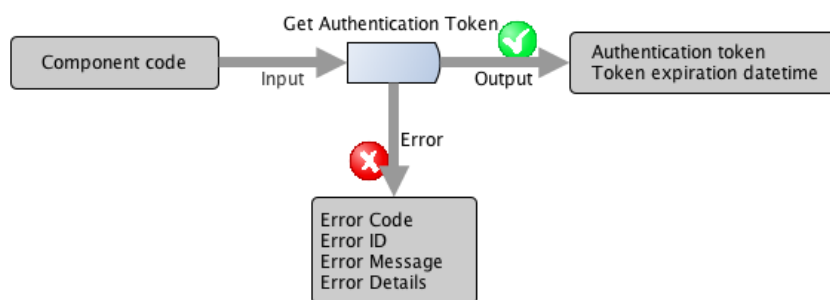
1683 **Figure 27 – Node interface – Overview**

1684 **5.3.2 Authentication service**

1685 There is one authentication service, named *GetAuthenticationToken*, used by a client
1686 component to retrieve a token supplied by a server component, which is referred as the
1687 “authentication-token”. The token has an expiration time (i.e. date and time). Such a token
1688 can be generated as a UUID.

1689 The client shall then return the authentication-token signed with the authentication certificate
1690 for every following request — see § 3.5.3. The client has to renew the authentication-token
1691 using the same service when expired.

1692 Figure 28 shows the node interface for the authentication service.



1693

1694

Figure 28 – Node interface – Authentication service

1695 The service request elements are provided in Table 33.

1696

Table 33 – Authentication – Service request elements

| Element name | Type | Description | Required |
|-----------------|---------|---|----------|
| componentCode | string | The component ID of the connecting client requesting for an authentication-token. | True |
| serviceMversion | integer | The MADES version of the current service that is requested by the client – see § 4.1.3. | True |

1697 The service response elements are provided in Table 34.

1698

Table 34 – Authentication – Service response elements

| Element name | Type | Description | Required |
|--------------|-----------|--|----------|
| authToken | string | The requested authentication token. | True |
| expiration | timestamp | The expiration date and time of the provided authentication-token. | True |

1699 **5.3.3 Messaging Services**

1700 **5.3.3.1 General**

1701 Messaging services are operations for bulk upload and download of messages.

1702 The download process is a two-phase operation: first the client downloads messages from
1703 server; then it confirms that the download was successful — see § 3.8.

1704 Two limits shall be configurable within each source component regarding the bulk transfer
1705 mechanism (defined by the network governance):

- 1706 • the maximum number of messages in one transfer.
- 1707 • the maximum allowed size for the request (upload) or the reply (download), which contains
1708 the messages.

1709 **5.3.3.2 “Transfer confirmation” versus “acceptance”**

1710 The transfer confirmation is a technical mechanism to notify a source component that the
1711 target component has taken responsibility for the message. If the source component does not

1712 receive the confirmation, it remains responsible for the message delivery and shall then
1713 transfer it again.

1714 The acceptance of a message by a component generally means more, i.e. that the message
1715 has passed additional validation checks. Moreover, acceptance always leads to an event
1716 notification (whether delivery or failure).

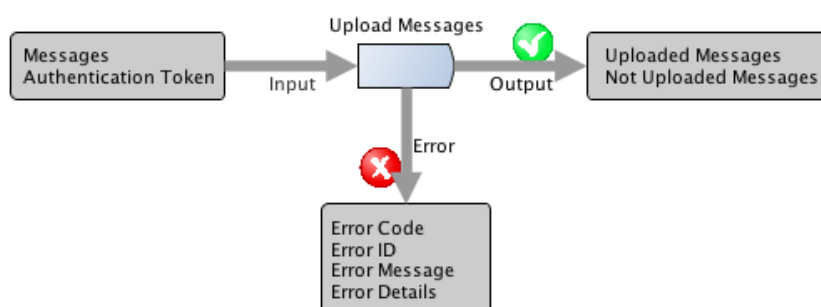
1717 Upload: When a node accepts an uploaded message, it delegates the event notification of the
1718 event n°3 to the sender's endpoint and uses the upload response to do so. Other components
1719 shall not use the upload response to reject a business-message. The interpretation of the
1720 possible responses by the sender's endpoint is:

- 1721 1. No confirmation is received; the message shall be transferred again.
- 1722 2. The message is accepted; a delivery acknowledgement shall be issued if the message
1723 is a business-message.
- 1724 3. The message is rejected (it can only on a business-message); a failure
1725 acknowledgement shall be issued and the message shall not be transferred again.

1726 Download: The target component of a download transfer (recipient' endpoint) always issues
1727 the acknowledgement. So there is no need to accept or reject a message when confirming the
1728 transfer (see *ConfirmDownload* § 5.3.3.5).

1729 5.3.3.3 UploadMessages service

1730 Figure 29 shows the node interface for the UploadMessages service:



1731

1732 **Figure 29 – Node interface – Messaging services – UploadMessages service**

1733 The service request elements are provided in Table 35.

1734 **Table 35 – UploadMessages – Service request elements**

| Element name | Element type | Description | Required |
|-----------------|---------------------------------------|---|----------|
| messages | InternalMessage[] (see Table 61) | The collection of the messages to be uploaded ordered according to priority rule in the client. | True |
| authToken | AuthenticationToken (see Table 53) | The authentication token provided by the server which is signed back by the client using the <u>authentication</u> certificate. | True |
| serviceMversion | integer | The MADES version of the current service that is requested by the client – see § 4.1.3. | True |

1735 The service response elements are provided in Table 36.

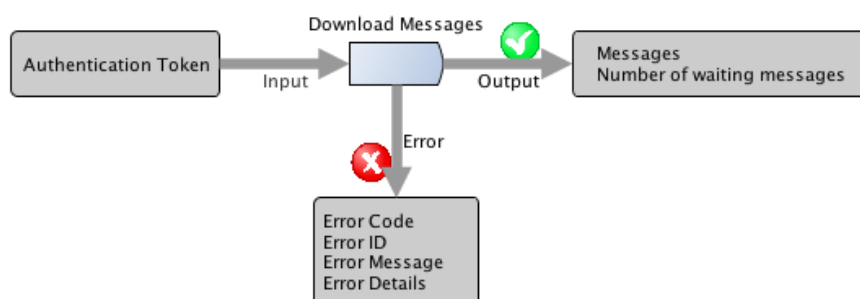
1736 **Table 36 – UploadMessages – Service response elements**

| Element name | Element type | Description | Required |
|---------------------|--|--|----------|
| uploadedMessages | string[] | The collection of the IDs of the messages which are confirmed as transferred, or accepted. | False |
| notUploadedMessages | notUploadedMessageResponse[] (see Table 73) | The collection of {ID, error details} for each non-accepted (i.e. rejected) message. | False |

1737 The ID of every message of the request shall belong to a collection of the response.

1738 **5.3.3.4 DownloadMessages service**

1739 Figure 30 shows the node interface for the DownloadMessages service:



1740

1741 **Figure 30 – Node interface – Messaging services – DownloadMessages service**

1742 A client component shall present the signed component ID of the endpoint for which it
1743 requests messages. The certificate used for signing the endpoint ID shall be the
1744 authentication certificate of the endpoint.

1745 The node shall verify that the endpoint ID is successfully signed with a non-revoked
1746 authentication certificate of the endpoint, and log an error message when the verification fails.

1747 The service request elements are provided in Table 37.

1748 **Table 37 – DownloadMessages – Service request elements**

| Element name | Element type | Description | Required |
|-----------------|---------------------------------------|---|----------|
| endpoints | Endpoint[] (see Table 60) | A one-element collection which contains the component ID of the recipient's endpoint whose messages are requested for download. | True |
| authToken | AuthenticationToken (see Table 53) | The authentication token provided by the server which is signed back by the client using the <u>authentication</u> certificate. | True |
| serviceMversion | integer | The MADES version of the current service that is requested by the client – see § 4.1.3. | True |

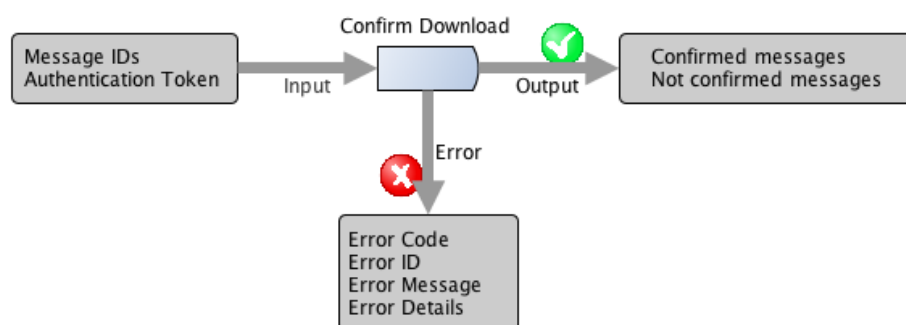
1749 The service response elements are provided in Table 38.

1750 **Table 38 – DownloadMessages – Service response elements**

| Element name | Element type | Description | Required |
|-----------------|-------------------------------------|--|----------|
| messages | InternalMessage[] (see Table 61) | The collection of the downloaded messages ordered according to priority rule in the server— see § 3.8 | False |
| waitingMessages | integer | The number of messages, matching the request, but not included in the current response and still waiting to be downloaded by the client. | True |

1751 **5.3.3.5 ConfirmDownload service**

1752 Figure 31 shows the node interface for the ConfirmDownloadMessages service:



1753
1754 **Figure 31 – Node interface – Messaging services – ConfirmDownload service**

1755 The client (source) component confirms the transfer of all or none of the messages that it
1756 previously received using a download request.

1757 The service request elements are provided in Table 39.

1758 **Table 39 – ConfirmDownload – Service request elements**

| Element name | Element type | Description | Required |
|-----------------|---------------------------------------|---|----------|
| messageIDs | string[] | The collection of the IDs of the messages whose download transfer is confirmed. | False |
| authToken | AuthenticationToken (see Table 53) | The authentication token provided by the server which is signed back by the client using the <u>authentication</u> certificate. | True |
| serviceMversion | integer | The MADES version of the current service that is requested by the client – see § 4.1.3. | True |

1759 The service response elements are provided in Table 40.

1760 **Table 40 – ConfirmDownload – Service response elements**

| Element name | Element type | Description | Required |
|----------------------|--|-------------|----------|
| confirmedMessages | string[] | (Unused) | False |
| notConfirmedMessages | NotConfirmedMessageResponse [] (see Table 72) | (Unused) | False |

1761 **5.3.4 Directory services**

1762 **5.3.4.1 SetComponentMversion Service**

1763 *SetComponentMversion* is used by a component to be accepted in the network — see § 4.2.2.

1764 To prevent that a component sends wrong data which could disrupt the network behaviour,
1765 the component ID shall be signed.

1766 The service request elements are provided in Table 41.

1767 **Table 41 – SetComponentMversion – Service request elements**

| Element name | Element type | Description | Required |
|-------------------|---------------------------------------|--|----------|
| componentCode | string | The ID of the component requesting for network acceptance. | True |
| signature | string | The RSA encoding of the SHA-1 hash of the component ID (componentCode) of the component requesting for network acceptance. The certificate used for encoding shall be the <u>authentication</u> certificate of the component. | True |
| certificateID | string | The ID of the certificate used to sign, i.e. to generate the "signature". | True |
| componentMVersion | integer | The installed MADES version of the component requesting for network acceptance. | True |
| authToken | AuthenticationToken (see Table 53) | The authentication token provided by the server which is signed back by the client using the <u>authentication</u> certificate. | True |
| serviceMversion | integer | The MADES version of the current service that is requested by the client – see § 4.1.3. | True |

1768 The service response elements are provided in Table 42.

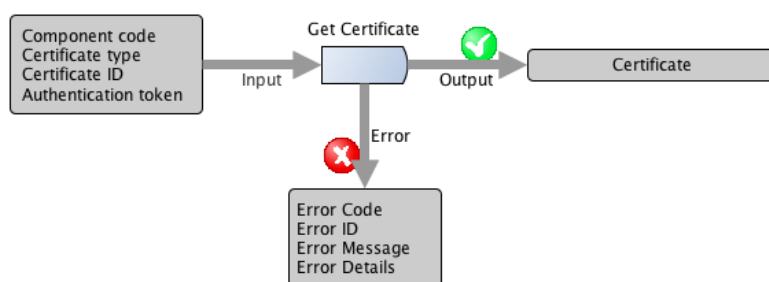
1769 **Table 42 – SetComponentMversion – Service response elements**

| Element name | Element type | Description | Required |
|--------------|--------------|--|----------|
| nodeMversion | integer | The installed MADES version of the home node. | True |
| acceptance | boolean | True if the component is accepted in the network – see § 4.2.2 | True |

1770 **5.3.4.2 GetCertificate service**

1771 *GetCertificate* is used to retrieve a certificate of a given type (signing, encryption, or
1772 authentication), owned by the given endpoint and possibly having the requested ID.

1773 Figure 32 shows the node interface for the GetCertificate service:



1774

1775 **Figure 32 – Node interface – Directory services – GetCertificate service**

1776 The service request elements are provided in Table 43.

1777 **Table 43 – GetCertificate – Service request elements**

| Element name | Element type | Description | Required |
|-----------------|---------------------------------------|---|----------|
| componentCode | string | The ID of the component that owns the requested certificate. | True |
| type | CertificateType (see Table 54) | The type of the requested certificate, | True |
| certificateID | string | The ID of the requested certificate. | False |
| authToken | AuthenticationToken (see Table 53) | The authentication token provided by the server which is signed back by the client using the <u>authentication</u> certificate. | True |
| serviceMversion | integer | The MADES version of the current service that is requested by the client – see § 4.1.3. | True |

1778 The service response elements are provided in Table 44.

1779 **Table 44 – GetCertificate - Service response elements**

| Element name | Element type | Description | Required |
|--------------|-------------------------------|---|----------|
| certificate | Certificate (See Table 54) | The returned certificate shall match the requested “type” and shall be owned by <i>componentCode</i> . If <i>certificateID</i> is also requested, the returned certificate shall also match the ID. Additional conditions about the validity and the revocation of the certificate are provided further. If no certificate matches, the response is empty. | False |

1780 Additional conditions are provided in Table 45.

1781 **Table 45 – GetCertificate – Additional conditions**

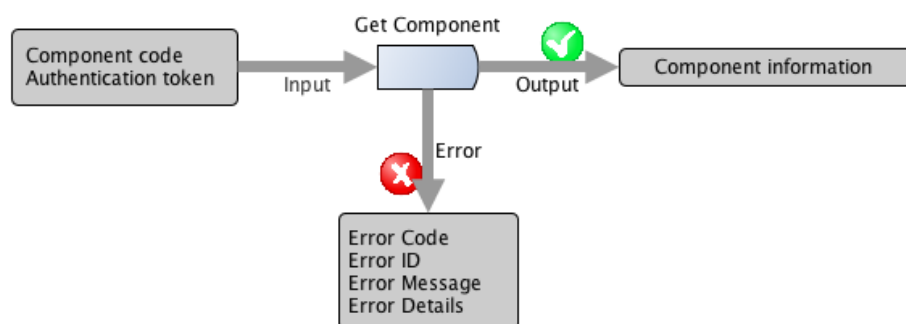
| Certificate type | When <i>certificateID</i> is requested | When <i>certificateID</i> is not requested |
|------------------|--|--|
| Authentication | The returned certificate shall be valid (not expired) and not revoked. | <u>Request Error</u> : this situation should never occur. A component shall only request for an authentication certificate to check a signed token or a signed component ID, and thus always knowing the certificate ID. |

| Certificate type | When <i>certificateID</i> is requested | When <i>certificateID</i> is not requested |
|------------------|--|---|
| Encryption | The returned certificate can be expired or revoked, for it may be requested to decrypt a message that was composed before the expiration time or the revocation time. | The returned certificate shall be valid and not revoked. When several certificates match conditions, the service shall return the certificate that expires first. |
| Signing | The returned certificate can be expired or revoked, for it may be requested to check the signature of a message that was composed before the expiration time or the revocation time. | <u>Request Error</u> : this situation should never occur. A component shall only request for a signing certificate to check a signature and thus always knowing the certificate ID. |

1782 **5.3.4.3 GetComponent service**

1783 *GetComponent* is used for retrieving descriptive and routing information on a component.

1784 Figure 33 shows the node interface for the GetComponent service:



1785

1786 **Figure 33 – Node interface – Directory services – GetComponent service**

1787 The service request elements are provided in Table 46.

1788 **Table 46 – GetComponent – Service request elements**

| Element name | Element type | Description | Required |
|-----------------|------------------------------------|---|----------|
| componentCode | string | The ID (or code) of the requested component. | True |
| authToken | AuthenticationToken (see Table 53) | The authentication token provided by the server which is signed back by the client using the <u>authentication</u> certificate. | True |
| serviceMversion | integer | The MADES version of the current service that is requested by the client – see § 4.1.3. | True |

1789 The service response elements are provided in Table 47.

1790 **Table 47 – GetComponent – Service response elements**

| Element name | Element type | Description | Required |
|--------------|-------------------------------------|--|----------|
| component | ComponentInformation (see Table 58) | The directory Information about the component. If the requested component does not exist, the response is empty. | False |

1791 **5.3.5 Node Synchronization interface**

1792 **5.3.5.1 GetNodeMversion service**

1793 The *GetNodeMversion* service is used by a node to get the Mversion of another node — see
1794 § 4.2.1.

1795 The service request elements are provided in Table 48.

1796 **Table 48 – GetNodeMversion – Service request elements**

| Element name | Element type | Description | Required |
|--------------|--------------|--|----------|
| mversion | integer | The installed MADES version of the requesting client node. | True |

1797 The service response elements are provided in Table 49.

1798 **Table 49 – GetNodeMversion – Service response elements**

| Element name | Element type | Description | Required |
|--------------|--------------|--|----------|
| mversion | integer | The installed MADES version of the replying server node. | True |
| nodeCode | string | The component ID of the replying server node. | True |

1799 **5.3.5.2 GetAllDirectoryData service**

1800 The *GetAllDirectoryData* service is used by the nodes to synchronize each other.

1801 The service request elements are provided in Table 50.

1802 **Table 50 – GetAllDirectoryData – Service request elements**

| Element name | Element type | Description | Required |
|-----------------|--------------|---|----------|
| dversion | integer | The version of the directory data of the server node that the client node already owns. NOTE No version shall be provided if the client synchronizes for the first time with the server. | False |
| serviceMversion | integer | The MADES version of the current service that is requested by the client – see § 4.1.3. | True |

1803 The service response elements are provided in Table 51.

1804 **Table 51 – GetAllDirectoryData – Service response elements**

| Element name | Element type | Description | Required |
|--------------|--------------|--|----------|
| dversion | integer | The current version of the directory reference data of the replying server node. | True |
| nodeCode | string | The component ID of the replying server node. | True |

| Element name | Element type | Description | Required |
|--------------|--|--|----------|
| components | ComponentDescription[] (see Table 57) | The collection of the descriptions of all components registered to the replying server node, plus the description of the node itself. The collection shall only be provided when the current directory version of the server node is strictly higher than the version already owned by the client node. | False |

1805 **5.4 Format of the node-list file**

1806 The node-list file shall be in UTF-8 encoding.

1807 The records in the file shall be delimited by the new line character (LF U+000A).

1808 Each record provides a list of attributes for one node. Attributes are delimited by the empty
1809 space character (SPACE U+0020) and shall appear in the order given in Table 52:

1810 **Table 52 – Node attributes ordered list**

| Attribute | Description | Required |
|--------------------|---|----------|
| Node component ID | The component ID of the node | True |
| Primary node URL | The primary URL to access the node, formatted according to RFC 1738 | True |
| Secondary node URL | The secondary URL to access the node, formatted according to RFC 1738 | False |

1811 **5.5 Typed Elements used by the interfaces**

1812 MTOM (*Message Transmission Optimization Mechanism*) is a W3C recommendation for
1813 handling binary data in SOAP messages — <http://www.w3.org/TR/soap12-mtom/>

1814 MTOM shall be used to optimise the size of the messages sent. Binary data in the SOAP
1815 message have to be encoded as text because SOAP uses XML. The base64 text encoding
1816 increases the size of the data by about 33 %. MTOM provides a way to send the binary data
1817 in the original binary form. MTOM optimizes the element content that is in the canonical
1818 lexical representation of the *xsd:base64Binary* type.

1819 All element types used in Clause 5 in the interfaces of the services are gathered from Table
1820 53 to Table 76.

1821 **Table 53 – AuthenticationToken**

| Element name | Element type | Description | Required |
|---------------|--------------|---|----------|
| token | string | The token received by the client (or requesting) component when it authenticated against the server (see the <i>GetAuthenticationToken</i> service). | True |
| signature | string | The signed token, i.e. the RSA encoding of the SHA-1 hash of the token. The encoding certificate shall be the authentication certificate of the client. | True |
| certificateID | string | The ID of the certificate used for signing the token. | True |

1822

Table 54 – Certificate

| Element name | Element type | Description | Required |
|---------------|--------------|--|----------|
| certificateID | string | The ID of the certificate. | True |
| certificate | base64Binary | The binary data of the certificate in DER (Distinguished Encoding Rules) format. | True |
| expiration | timestamp | The cache expiration date-time of the certificate if cached by client — see § 3.16.5. <i>Do not confuse with the expiration date of the certificate as defined by the certificate issuer and included in the certificate itself.</i> <i>The element is required for directory service, but not for directory synchronization</i> | |

1823

Table 55 – CertificateType – string enumeration

| String value | Description |
|----------------|---|
| AUTHENTICATION | A certificate used for TLS and token authentication |
| ENCRYPTION | A certificate used for encryption |
| SIGNING | A certificate used for signing |

1824

Table 56 – ComponentCertificate

| Element name | Element type | Description | Required |
|--------------|--------------------------------|--|----------|
| type | CertificateType (see Table 55) | The type of the certificate (e.g. encryption, signing, authentication) | True |
| revoked | boolean | 'true' if the certificate has been revoked. | False |
| certificate | Certificate (see Table 54) | The certificate. | True |

1825

Table 57 – ComponentDescription

| Element name | Element type | Description | Required |
|--------------|---------------------------------------|---|----------|
| information | ComponentInformation (see Table 58) | All about the component: ID, type, contact information, routing information. | True |
| certificates | ComponentCertificate[] (see Table 56) | The collection of the certificates owned by the component, whatever type (signing, encryption and authentication), and possibly more than one for some types. | True |

1826

Table 58 – ComponentInformation

| Element name | Element type | Description | Required |
|--------------|------------------------------|--|----------|
| code | string | The component ID to which is associated all information of the current data structure. | True |
| type | ComponentType (see Table 59) | The type of component. | True |
| organization | string | The organization responsible for the component. | True |
| person | string | The technical contact person. | True |
| email | string | The email of the technical contact person. | True |
| phone | string | The phone number of the technical contact person. | True |

| Element name | Element type | Description | Required |
|--------------|-----------------------------------|---|----------|
| routing | RoutingInformation (see Table 75) | The routing information to access to the component (ex: URLs) | True |
| expiration | timestamp | The cache expiration date-time of the information if cached by client — see § 3.16.5. <i>The element is required for directory service, but not for directory synchronization</i> | |
| codeMversion | integer | The MADES version to which the component complies – see § 4.1.2. <i>NOTE Information should be initialized in the home node at registration time. Otherwise it remains unknown until the component first connects to the network, and no message can be sent to the component.</i> | False |

1827 **Table 59 – ComponentType – string enumeration**

| String value | Description |
|--------------|-------------------------------|
| NODE | The component is a node. |
| ENDPOINT | The component is an endpoint. |

1828 **Table 60 – Endpoint**

| Element name | Element type | Description | Required |
|---------------|--------------|---|----------|
| code | string | The component ID of an endpoint. | True |
| signature | string | The RSA encoding of the SHA-1 hash of the component ID of the endpoint. The certificate used for encoding is the <u>authentication</u> certificate of the endpoint. | True |
| certificateID | string | The ID of the certificate used to encode the signature. | True |

1829 **Table 61 – InternalMessage**

| Element name | Element type | Description | Required |
|---------------------|--------------|---|----------|
| <u>messageID</u> | string | The ID of the message. | True |
| <u>receiverCode</u> | string | <i>Business-message, Tracing-message</i> → The component ID of the recipient's endpoint. <i>Acknowledgement</i> → The <i>senderCode</i> of the original message. | True |
| <u>businessType</u> | string | <i>Business-message</i> → The business-type as provided by the sender's BA. <i>Tracing-message</i> → Irrelevant, but at least one character needed. <i>Acknowledgement</i> → The business-type of the original message. | True |
| <u>content</u> | base64Binary | <i>Business-message</i> → The encrypted content of the message, possibly compressed before encrypted. <i>Tracing-message</i> → A non empty (at least one character) irrelevant and not compressed but encrypted content. <i>Acknowledgement</i> → see § 3.10.4. | True |

| Element name | Element type | Description | Required |
|--------------------------|---------------------------------------|--|----------|
| <u>extension</u> | string | <i>Business-message</i> → The file extension for the document — only used if the content was transferred to the sender's endpoint through a file and by the FSSF interface (see § 5.2.3). <i>Acknowledgement, Tracing-message</i> → Not used. | False |
| <u>generated</u> | dateTime | <i>Business-message, Tracing-message</i> → The date and time when the message was created by the sender's endpoint. <i>Acknowledgement</i> → The date and time of the notified event, i.e. when the acknowledgement was created in the sending component. | True |
| expirationTime | timestamp | <i>Business-message, Tracing-message</i> → The expiration date and time of the message — set by the sender's endpoint when accepting the message (see § 3.9). <i>Acknowledgement</i> → The <i>expirationTime</i> of the original message. | True |
| <u>senderCode</u> | string | <i>Business-message, Tracing-message</i> → The component ID of the sender's endpoint. <i>Acknowledgement</i> → The ID of the component sending the acknowledgement. | True |
| <u>senderDescription</u> | string | The display name of the <i>senderCode</i> component. | True |
| <u>internalType</u> | InternalMessageType (see Table 62) | The technical type of the message. | True |
| <u>relatedMessageID</u> | string | <i>Business-message, Tracing-message</i> → Not used. <i>Acknowledgement</i> → The message ID of the original message.. | False |
| <u>SenderApplication</u> | string | <i>Business-message, Tracing-message</i> → The identifier of the sender's BA as provided when sending the document. <i>Acknowledgement</i> → Not used. | False |
| <u>baMessageID</u> | string | <i>Business-message, Tracing-message</i> → An identifier of the document as provided by the sender's BA. <i>Acknowledgement</i> → Not used. | False |
| metadata | MessageMetadata (see Table 63) | The metadata added to the message by compression, signature or encryption – see § 3.14. | False |
| messageMversion | integer | The MADES version to which the message complies – see § 4.1.4 | True |

1830 NOTE The underlined attributes are those included in the manifest used to generate the message signature – see
1831 § 3.14.3.

1832 The “message header” refers to the set of all elements except “content”.

1833 **Table 62 – InternalMessageType – string enumeration**

| String Value | Description |
|--------------------------|---|
| STANDARD_MESSAGE | A business-message but not a tracing-message. |
| DELIVERY_ACKNOWLEDGEMENT | An acknowledgement notifying that the original STANDARD_MESSAGE has been accepted by a component. |
| RECEIVE_ACKNOWLEDGEMENT | An acknowledgement notifying that the original STANDARD_MESSAGE has been transferred to a recipient's BA. |

| String Value | Description |
|-------------------------|--|
| FAILURE_ACKNOWLEDGEMENT | A failure-acknowledgement. |
| TRACING_MESSAGE | A tracing-message – see § 3.12. |
| TRACING_ACKNOWLEDGEMENT | An acknowledgement notifying that the original TRACING_MESSAGE has been accepted by a component. |

1834 **Table 63 – MessageMetadata**

| Element name | Element type | Description | Required |
|-------------------|--------------------|--|----------|
| messageProcessors | MessageProcessor[] | A collection of metadata, each from a used message processor (collection count may range from 1 to 3). | False |

1835 **Table 64 – MessageProcessor**

| Element name | Element type | Description | Required |
|---------------|--------------|--|----------|
| processorID | string | The unique ID of the message processor. There are 3 processors whose IDs are: “signature” “encryption” “compressor” | True |
| processorData | Map | A collection of named values. | True |

1836 **Table 65 – Map**

| Element name | Element type | Description | Required |
|--------------|--------------|--|----------|
| entries | MapEntry[] | A collection of data, each provided with a name and a value, i.e. a set composed of a key (name), a type (format) and a value (according to the type). | False |

1837 **Table 66 – MapEntry**

| Element name | Element type | Description | Required |
|--------------|--------------|----------------------------------|----------|
| key | string | The name of the metadata | True |
| type | ValueType | The type/format of the metadata. | True |
| value | string | The value of the metadata. | True |

1838 **Table 67 – ValueType (enumeration)**

| String Value | Description |
|--------------|--|
| STRING | String |
| LONG | A 64bit number expressed as string. Example number 42 is represented as string "42" (without quotes) |
| BYTE_ARRAY | ↔ base64Binary type |
| BOOLEAN | A string equal to “true” or “false”. |

1839

Table 68 – MessageState (string enumeration)

| String value | Description |
|--------------|---|
| VERIFYING | The acceptance of the message by the sender's endpoint is pending due to connectivity problem between the sender's endpoint and the directory services. |
| ACCEPTED | The message has been accepted by the sender's endpoint. |
| TRANSPORTED | The message has been accepted by an intermediate component (except the recipient's endpoint or a recipient's BA). |
| DELIVERED | The message has been accepted by the recipient's endpoint. |
| RECEIVED | The message has been accepted by a recipient's BA. |
| FAILED | The processing of the message has failed and the delivery is stopped. |

1840

Table 69 – MessageStatus

| Element name | Element type | Description |
|-------------------|------------------------------------|--|
| messageID | string | The UUID (Universal Unique ID) of the message whose status is reported in this data structure — see § 3.2. |
| state | MessageState (see Table 68) | The delivery-status of the requested message. (see values in § 3.4 - uppercase) |
| receiverCode | string | The component ID of the recipient's endpoint of the message. |
| senderCode | string | The component ID of the sender's endpoint of the message. |
| businessType | string | The business-type of the message. |
| senderApplication | string | The identifier of sender's BA, if any and as provided by the sender's BA in the <i>SendMessage</i> service. |
| baMessageID | string | The identifier of the message assigned by the sending BA, if any and as provided by the sender's BA in the <i>SendMessage</i> service. |
| sendTimestamp | dateTime | The time when the message was created by the sender's endpoint (The <i>generated</i> element of the <i>InternalMessage</i> type – see Table 61). |
| receiveTimestamp | dateTime | The "reception time" of the message in the sender's endpoint. It is the time when the message state was set to DELIVERED in the sender's endpoint, which is also the time when the acknowledgement with the DELIVERED status (event n°6) was sent. |
| trace | MessageTraceltem [] (see Table 70) | The collection of the traces reporting the events about the message delivery. |

1841

Table 70 – MessageTraceltem

| Element name | Element type | Description | Required |
|-----------------------|----------------------------------|---|----------|
| timestamp | dateTime | The date and time of the reported event. | True |
| state | MessageTraceState (see Table 71) | The reported event | True |
| component | string | The ID of the component (see § 3.2) where the event happened. | True |
| Component description | string | The display name of the component where the event happened. | True |
| Details | string | The English readable details about the event. | False |

1842

Table 71 – MessageTraceState (string enumeration)

| String value | Description |
|--------------|---|
| VERIFYING | The acceptance of the message by the sender's endpoint is pending due to connectivity problem between the sender's endpoint and the directory services. <i>(internal event reported by the sender's endpoint).</i> |
| ACCEPTED | The message has been accepted by the sender's endpoint. <i>(internal event reported by the sender's endpoint).</i> |
| TRANSPORTED | The message has been accepted by an intermediate component (except the recipient's endpoint or a recipient's BA). <i>(event reported using a DELIVERY_ACKNOWLEDGEMENT or a TRACING_ACKNOWLEDGEMENT – see Table 62)</i> |
| DELIVERED | The message has been accepted by the recipient's endpoint. <i>(event reported using a DELIVERY_ACKNOWLEDGEMENT or a TRACING_ACKNOWLEDGEMENT – see Table 62)</i> |
| RECEIVED | The message has been accepted by a recipient's BA. <i>(event reported using an RECEIVE_ACKNOWLEDGEMENT – see Table 62)</i> |
| FAILED | The processing of the message has failed and the delivery is stopped. <i>(internal event reported by the sender's endpoint or event reported using a FAILURE_ACKNOWLEDGEMENT – see Table 62)</i> |

1843

Table 72 – NotConfirmedMessageResponse

| Element name | Element type | Description | Required |
|--------------|--------------|--|----------|
| messageID | string | The ID of the message whose upload failed. | True |
| errorCode | string | A code representing the type of the error (e.g. validation error, unexpected error) | True |
| errorID | string | Unique identification of the error and for the component implementation. The error ID shall be always written in the logs. | True |
| errorMessage | string | An English readable text describing the error. | True |
| errorDetails | string | (optional) Additional English readable details about the error context. | False |

1844

Table 73 – NotUploadedMessageResponse

| Element name | Element type | Description | Required |
|----------------------|--------------|--|----------|
| messageID | string | The ID of the message whose upload failed. | True |
| fatal | boolean | Set to true if the error is not recoverable. The message shall then be set in the failed state by the client source component, which shall never try to upload it again. | True |
| businessErrorMessage | string | An English readable description of the error. | False |
| errorCode | string | A code representing the type of the error (e.g. validation error, unexpected error) | True |
| errorID | string | Unique identification of the error and for the component implementation. The error ID shall be always written in the logs. | True |

| Element name | Element type | Description | Required |
|--------------|--------------|---|----------|
| errorMessage | string | An English readable text describing the error. | True |
| errorDetails | string | (optional) Additional English readable details about the error context. | False |

1845

Table 74 – ReceivedMessage

| Element name | Element type | Description |
|-------------------|--------------|--|
| messageID | string | The UUID (Universal Unique ID) of the message — see § 3.2. |
| receiverCode | string | The component ID of the recipient's endpoint of the message — see § 3.2. |
| senderCode | string | The component ID of the sender's endpoint of the message — see § 3.2. |
| businessType | string | The business-type of the message currently transferred to the BA. |
| content | base64Binary | The content of the message as provided by the sender's BA in the <i>SendMessage</i> service. |
| senderApplication | string | The identifier of sender's BA, if any and as provided by the sender's BA in the <i>SendMessage</i> service. |
| baMessageID | string | The identifier of the message assigned by the sending BA, if any and as provided by the sender's BA in the <i>SendMessage</i> service. |

1846

Table 75 – RoutingInformation

| Element name | Element type | Description | Required |
|--------------|--------------|--|----------|
| node | string | The component ID of the component's home node. | True |
| primaryURL | string | The primary URL of the node according to RFC 1738 | True |
| secondaryURL | string | The secondary URL of the node according to RFC 1738 | False |
| nodeMversion | integer | The installed MADES version of the component's home node – see § 4.1.2 | True |

1847

Table 76 – SentMessage

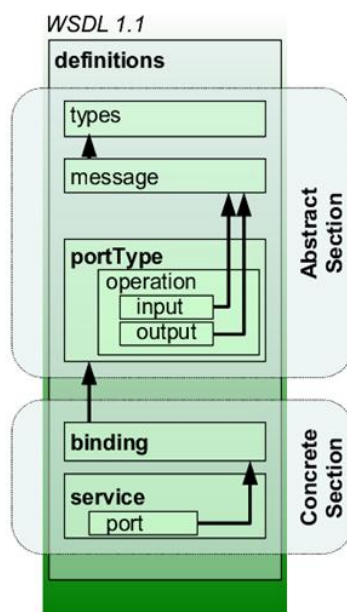
| Element name | Element type | Description | Required |
|-------------------|--------------|---|----------|
| receiverCode | string | The component ID of the recipient's endpoint (see § 3.2) — Pattern: [A-Za-z0-9-@]+ | True |
| businessType | string | The business-type of the message (see § 3.3) — Pattern: [A-Za-z0-9]+ | True |
| baMessageID | string | An identifier of the document provided by the sender's BA. Information is transported "as is" to the recipient's BA — Pattern: [A-Za-z0-9]* | False |
| senderApplication | string | The identifier of the sender's BA. Information is transported "as is" to the recipient's BA — Pattern: [A-Za-z0-9]* | False |
| content | base64Binary | The content of the message, i.e. the business document. NOTE There is no constraint about the structure of the document which is processed as a stream of bytes. E.g. it can be a human-readable XML document, multiples files compressed in ZIP format. | True |

1848 5.6 Description of the services

1849 5.6.1 About WSDL and SOAP

1850 The services are described using the Web Services Description Language (WSDL) 1.1¹¹.
1851 See <http://www.w3.org/TR/wsdl> and Figure 34

1852 The SOAP 1.1 and SOAP 1.2 bindings allow using the interfaces via SOAP 1.1 and SOAP 1.2
1853 protocols.



1854

Figure 34 – WSDL 1.1 definitions

1855

1856 5.6.2 Endpoint interface

```

1857 <?xml version="1.0" encoding="UTF-8"?>
1858 <wSDL:definitions name="MadesEndpoint" targetNamespace="http://mades.entsoe.eu/"
1859 xmlns:wSDL="http://schemas.xmlsoap.org/wSDL/"
1860 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
1861 xmlns:soap="http://schemas.xmlsoap.org/wSDL/soap/"
1862 xmlns:eCP="http://mades.entsoe.eu/"
1863 xmlns:soap12="http://schemas.xmlsoap.org/wSDL/soap12/">
1864
1865   <wSDL:types>
1866     <xsd:schema targetNamespace="http://mades.entsoe.eu/">
1867
1868       <xsd:element name="SendMessageRequest">
1869         <xsd:complexType>
1870           <xsd:sequence>
1871             <xsd:element name="message" type="mades:SendMessage"/>
1872             <xsd:element minOccurs="0" name="conversationID" nillable="true"
1873 type="xsd:string"/>
1874           </xsd:sequence>
1875         </xsd:complexType>
1876       </xsd:element>
1877
1878       <xsd:element name="SendMessageResponse">
1879         <xsd:complexType>

```

¹¹ Figure 34 from http://en.wikipedia.org/wiki/Web_Services_Description_Language.

```

1880         <xsd:sequence>
1881         <xsd:element name="messageID" type="xsd:string"/>
1882         </xsd:sequence>
1883     </xsd:complexType>
1884 </xsd:element>
1885
1886 <xsd:element name="SendMessageError">
1887     <xsd:complexType>
1888         <xsd:sequence>
1889             <xsd:element name="errorCode" type="xsd:string"/>
1890             <xsd:element name="errorID" type="xsd:string"/>
1891             <xsd:element name="errorMessage" type="xsd:string"/>
1892             <xsd:element minOccurs="0" name="receiverCode" nillable="true"
1893 type="xsd:string"/>
1894             <xsd:element minOccurs="0" name="errorDetails" type="xsd:string"/>
1895         </xsd:sequence>
1896     </xsd:complexType>
1897 </xsd:element>
1898
1899 <xsd:element name="ReceiveMessageRequest">
1900     <xsd:complexType>
1901         <xsd:sequence>
1902             <xsd:element name="businessType" type="xsd:string"/>
1903             <xsd:element name="downloadMessage" type="xsd:boolean"/>
1904         </xsd:sequence>
1905     </xsd:complexType>
1906 </xsd:element>
1907
1908 <xsd:element name="ReceiveMessageResponse">
1909     <xsd:complexType>
1910         <xsd:sequence>
1911             <xsd:element minOccurs="0" name="receivedMessage" nillable="true"
1912 type="mades:ReceivedMessage"/>
1913             <xsd:element name="remainingMessagesCount" type="xsd:long"/>
1914         </xsd:sequence>
1915     </xsd:complexType>
1916 </xsd:element>
1917
1918 <xsd:element name="ReceiveMessageError">
1919     <xsd:complexType>
1920         <xsd:sequence>
1921             <xsd:element name="errorCode" type="xsd:string"/>
1922             <xsd:element name="errorID" type="xsd:string"/>
1923             <xsd:element name="errorMessage" type="xsd:string"/>
1924             <xsd:element minOccurs="0" name="businessType" nillable="true"
1925 type="xsd:string"/>
1926             <xsd:element minOccurs="0" name="errorDetails" type="xsd:string"/>
1927         </xsd:sequence>
1928     </xsd:complexType>
1929 </xsd:element>
1930
1931 <xsd:element name="ConfirmReceiveMessageRequest">
1932     <xsd:complexType>
1933         <xsd:sequence>
1934             <xsd:element name="messageID" type="xsd:string"/>
1935         </xsd:sequence>
1936     </xsd:complexType>
1937 </xsd:element>
1938
1939 <xsd:element name="ConfirmReceiveMessageResponse">
1940     <xsd:complexType>
1941         <xsd:sequence>
1942             <xsd:element name="messageID" type="xsd:string"/>
1943         </xsd:sequence>
1944     </xsd:complexType>
1945 </xsd:element>
1946

```

```

1947     <xsd:element name="ConfirmReceiveMessageError">
1948     <xsd:complexType>
1949     <xsd:sequence>
1950     <xsd:element name="errorCode" type="xsd:string"/>
1951     <xsd:element name="errorID" type="xsd:string"/>
1952     <xsd:element name="errorMessage" type="xsd:string"/>
1953     <xsd:element minOccurs="0" name="messageID" nillable="true"
1954 type="xsd:string"/>
1955     <xsd:element minOccurs="0" name="errorDetails" type="xsd:string"/>
1956     </xsd:sequence>
1957     </xsd:complexType>
1958 </xsd:element>
1959
1960     <xsd:complexType name="SentMessage">
1961     <xsd:sequence>
1962     <xsd:element name="receiverCode" type="xsd:string"/>
1963     <xsd:element name="businessType" type="xsd:string"/>
1964     <xsd:element name="content" type="xsd:base64Binary"/>
1965     <xsd:element minOccurs="0" name="senderApplication" nillable="true"
1966 type="xsd:string"/>
1967     <xsd:element minOccurs="0" name="baMessageID" nillable="true"
1968 type="xsd:string"/>
1969     </xsd:sequence>
1970 </xsd:complexType>
1971
1972     <xsd:complexType name="ReceivedMessage">
1973     <xsd:sequence>
1974     <xsd:element name="messageID" type="xsd:string"/>
1975     <xsd:element name="receiverCode" type="xsd:string"/>
1976     <xsd:element name="senderCode" type="xsd:string"/>
1977     <xsd:element name="businessType" type="xsd:string"/>
1978     <xsd:element name="content" type="xsd:base64Binary"/>
1979     <xsd:element minOccurs="0" name="senderApplication" nillable="true"
1980 type="xsd:string"/>
1981     <xsd:element minOccurs="0" name="baMessageID" nillable="true"
1982 type="xsd:string"/>
1983     </xsd:sequence>
1984 </xsd:complexType>
1985
1986     <xsd:complexType name="MessageStatus">
1987     <xsd:sequence>
1988     <xsd:element name="messageID" type="xsd:string"/>
1989     <xsd:element name="state" type="mades:MessageState"/>
1990     <xsd:element name="receiverCode" type="xsd:string"/>
1991     <xsd:element name="senderCode" type="xsd:string"/>
1992     <xsd:element name="businessType" type="xsd:string"/>
1993     <xsd:element minOccurs="0" name="senderApplication" nillable="true"
1994 type="xsd:string"/>
1995     <xsd:element minOccurs="0" name="baMessageID" nillable="true"
1996 type="xsd:string"/>
1997     <xsd:element name="sendTimestamp" type="xsd:dateTime"/>
1998     <xsd:element minOccurs="0" name="receiveTimestamp" nillable="true"
1999 type="xsd:dateTime"/>
2000     <xsd:element name="trace" nillable="true" type="mades:MessageTrace"/>
2001     </xsd:sequence>
2002 </xsd:complexType>
2003
2004     <xsd:complexType name="MessageTrace">
2005     <xsd:sequence>
2006     <xsd:element maxOccurs="unbounded" name="trace"
2007 type="mades:MessageTraceItem"/>
2008     </xsd:sequence>
2009 </xsd:complexType>
2010
2011     <xsd:complexType name="MessageTraceItem">
2012     <xsd:sequence>
2013     <xsd:element name="timestamp" type="xsd:dateTime"/>

```



```

2014         <xsd:element name="state" type="mades:MessageTraceState"/>
2015         <xsd:element name="component" type="xsd:string"/>
2016         <xsd:element name="componentDescription" type="xsd:string"/>
2017         <xsd:element name="details" nillable="true" type="xsd:string"/>
2018     </xsd:sequence>
2019 </xsd:complexType>
2020
2021     <xsd:element name="ConnectivityTestRequest">
2022         <xsd:complexType>
2023             <xsd:sequence>
2024                 <xsd:element name="receiverCode" type="xsd:string"/>
2025             </xsd:sequence>
2026         </xsd:complexType>
2027     </xsd:element>
2028
2029     <xsd:element name="ConnectivityTestResponse">
2030         <xsd:complexType>
2031             <xsd:sequence>
2032                 <xsd:element name="messageID" type="xsd:string"/>
2033             </xsd:sequence>
2034         </xsd:complexType>
2035     </xsd:element>
2036
2037     <xsd:element name="ConnectivityTestError">
2038         <xsd:complexType>
2039             <xsd:sequence>
2040                 <xsd:element name="errorCode" type="xsd:string"/>
2041                 <xsd:element name="errorID" type="xsd:string"/>
2042                 <xsd:element name="errorMessage" type="xsd:string"/>
2043                 <xsd:element minOccurs="0" name="receiverCode" nillable="true"
2044 type="xsd:string"/>
2045                 <xsd:element minOccurs="0" name="errorDetails" type="xsd:string"/>
2046             </xsd:sequence>
2047         </xsd:complexType>
2048     </xsd:element>
2049
2050     <xsd:element name="CheckMessageStatusRequest">
2051         <xsd:complexType>
2052             <xsd:sequence>
2053                 <xsd:element name="messageID" type="xsd:string"/>
2054             </xsd:sequence>
2055         </xsd:complexType>
2056     </xsd:element>
2057
2058     <xsd:element name="CheckMessageStatusResponse">
2059         <xsd:complexType>
2060             <xsd:sequence>
2061                 <xsd:element name="messageStatus" type="mades:MessageStatus"/>
2062             </xsd:sequence>
2063         </xsd:complexType>
2064     </xsd:element>
2065
2066     <xsd:element name="CheckMessageStatusError">
2067         <xsd:complexType>
2068             <xsd:sequence>
2069                 <xsd:element name="errorCode" type="xsd:string"/>
2070                 <xsd:element name="errorID" type="xsd:string"/>
2071                 <xsd:element name="errorMessage" type="xsd:string"/>
2072                 <xsd:element minOccurs="0" name="messageID" nillable="true"
2073 type="xsd:string"/>
2074                 <xsd:element minOccurs="0" name="errorDetails" type="xsd:string"/>
2075             </xsd:sequence>
2076         </xsd:complexType>
2077     </xsd:element>
2078
2079     <xsd:simpleType name="MessageState">
2080         <xsd:restriction base="xsd:string">

```

```

2081         <xsd:enumeration value="VERIFYING"/>
2082         <xsd:enumeration value="ACCEPTED"/>
2083         <xsd:enumeration value="DELIVERING"/>
2084         <xsd:enumeration value="DELIVERED"/>
2085         <xsd:enumeration value="RECEIVED"/>
2086         <xsd:enumeration value="FAILED"/>
2087     </xsd:restriction>
2088 </xsd:simpleType>
2089
2090     <xsd:simpleType name="MessageTraceState">
2091         <xsd:restriction base="xsd:string">
2092             <xsd:enumeration value="VERIFYING"/>
2093             <xsd:enumeration value="ACCEPTED"/>
2094             <xsd:enumeration value="TRANSPORTED"/>
2095             <xsd:enumeration value="DELIVERED"/>
2096             <xsd:enumeration value="RECEIVED"/>
2097             <xsd:enumeration value="FAILED"/>
2098         </xsd:restriction>
2099     </xsd:simpleType>
2100 </xsd:schema>
2101 </wsdl:types>
2102
2103 <wsdl:message name="SendMessageRequest">
2104     <wsdl:part name="parameters" element="mades:SendMessageRequest"/>
2105 </wsdl:message>
2106
2107 <wsdl:message name="SendMessageResponse">
2108     <wsdl:part name="parameters" element="mades:SendMessageResponse"/>
2109 </wsdl:message>
2110
2111 <wsdl:message name="ConnectivityTestFault">
2112     <wsdl:part name="fault" element="mades:ConnectivityTestError"/>
2113 </wsdl:message>
2114
2115 <wsdl:message name="ReceiveMessageRequest">
2116     <wsdl:part name="parameters" element="mades:ReceiveMessageRequest"/>
2117 </wsdl:message>
2118
2119 <wsdl:message name="ConfirmReceiveMessageRequest">
2120     <wsdl:part name="parameters" element="mades:ConfirmReceiveMessageRequest"/>
2121 </wsdl:message>
2122
2123 <wsdl:message name="ConnectivityTestRequest">
2124     <wsdl:part name="parameters" element="mades:ConnectivityTestRequest"/>
2125 </wsdl:message>
2126
2127 <wsdl:message name="CheckMessageStatusResponse">
2128     <wsdl:part name="parameters" element="mades:CheckMessageStatusResponse"/>
2129 </wsdl:message>
2130
2131 <wsdl:message name="ConfirmReceiveMessageResponse">
2132     <wsdl:part name="parameters" element="mades:ConfirmReceiveMessageResponse"/>
2133 </wsdl:message>
2134
2135 <wsdl:message name="ReceiveMessageFault">
2136     <wsdl:part name="fault" element="mades:ReceiveMessageError"/>
2137 </wsdl:message>
2138
2139 <wsdl:message name="CheckMessageStatusFault">
2140     <wsdl:part name="fault" element="mades:CheckMessageStatusError"/>
2141 </wsdl:message>
2142
2143 <wsdl:message name="CheckMessageStatusRequest">
2144     <wsdl:part name="parameters" element="mades:CheckMessageStatusRequest"/>
2145 </wsdl:message>
2146
2147 <wsdl:message name="ConfirmReceiveMessageFault">

```

```

2148     <wsdl:part name="fault" element="mades:ConfirmReceiveMessageError"/>
2149 </wsdl:message>
2150
2151 <wsdl:message name="SendMessageFault">
2152   <wsdl:part name="fault" element="mades:SendMessageError"/>
2153 </wsdl:message>
2154
2155 <wsdl:message name="ReceiveMessageResponse">
2156   <wsdl:part name="parameters" element="mades:ReceiveMessageResponse"/>
2157 </wsdl:message>
2158
2159 <wsdl:message name="ConnectivityTestResponse">
2160   <wsdl:part name="parameters" element="mades:ConnectivityTestResponse"/>
2161 </wsdl:message>
2162
2163 <wsdl:portType name="MadesEndpoint">
2164   <wsdl:operation name="SendMessage">
2165     <wsdl:input message="mades:SendMessageRequest"/>
2166     <wsdl:output message="mades:SendMessageResponse"/>
2167     <wsdl:fault name="SendMessageError" message="mades:SendMessageFault"/>
2168   </wsdl:operation>
2169   <wsdl:operation name="ReceiveMessage">
2170     <wsdl:input message="mades:ReceiveMessageRequest"/>
2171     <wsdl:output message="mades:ReceiveMessageResponse"/>
2172     <wsdl:fault name="ReceiveMessageError" message="mades:ReceiveMessageFault"/>
2173   </wsdl:operation>
2174   <wsdl:operation name="ConfirmReceiveMessage">
2175     <wsdl:input message="mades:ConfirmReceiveMessageRequest"/>
2176     <wsdl:output message="mades:ConfirmReceiveMessageResponse"/>
2177     <wsdl:fault name="ConfirmReceiveMessageError"
2178 message="mades:ConfirmReceiveMessageFault"/>
2179   </wsdl:operation>
2180   <wsdl:operation name="ConnectivityTest">
2181     <wsdl:input message="mades:ConnectivityTestRequest"/>
2182     <wsdl:output message="mades:ConnectivityTestResponse"/>
2183     <wsdl:fault name="ConnectivityTestError"
2184 message="mades:ConnectivityTestFault"/>
2185   </wsdl:operation>
2186   <wsdl:operation name="CheckMessageStatus">
2187     <wsdl:input message="mades:CheckMessageStatusRequest"/>
2188     <wsdl:output message="mades:CheckMessageStatusResponse"/>
2189     <wsdl:fault name="CheckMessageStatusError"
2190 message="mades:CheckMessageStatusFault"/>
2191   </wsdl:operation>
2192 </wsdl:portType>
2193
2194 <wsdl:binding name="MadesEndpointSOAP12" type="mades:MadesEndpoint">
2195   <soap12:binding style="document"
2196 transport="http://schemas.xmlsoap.org/soap/http"/>
2197   <wsdl:operation name="SendMessage">
2198     <soap12:operation soapAction="http://mades.entsoe.eu/SendMessage"/>
2199     <wsdl:input> <soap12:body use="literal"/> </wsdl:input>
2200     <wsdl:output> <soap12:body use="literal"/> </wsdl:output>
2201     <wsdl:fault name="SendMessageError"> <soap12:fault name="SendMessageError"
2202 use="literal"/> </wsdl:fault>
2203   </wsdl:operation>
2204   <wsdl:operation name="ReceiveMessage">
2205     <soap12:operation soapAction="http://mades.entsoe.eu/ReceiveMessage"/>
2206     <wsdl:input> <soap12:body use="literal"/> </wsdl:input>
2207     <wsdl:output> <soap12:body use="literal"/> </wsdl:output>
2208     <wsdl:fault name="ReceiveMessageError"> <soap12:fault
2209 name="ReceiveMessageError" use="literal"/> </wsdl:fault>
2210   </wsdl:operation>
2211   <wsdl:operation name="ConfirmReceiveMessage">
2212     <soap12:operation soapAction="http://mades.entsoe.eu/ConfirmReceiveMessage"/>
2213     <wsdl:input> <soap12:body use="literal"/> </wsdl:input>
2214     <wsdl:output> <soap12:body use="literal"/> </wsdl:output>

```

```

2215     <wsdl:fault name="ConfirmReceiveMessageError"> <soap12:fault
2216 name="ConfirmReceiveMessageError" use="literal"/> </wsdl:fault>
2217 </wsdl:operation>
2218 <wsdl:operation name="ConnectivityTest">
2219 <soap12:operation soapAction="http://mades.entsoe.eu/ConnectivityTest"/>
2220 <wsdl:input> <soap12:body use="literal"/> </wsdl:input>
2221 <wsdl:output> <soap12:body use="literal"/> </wsdl:output>
2222 <wsdl:fault name="ConnectivityTestError"> <soap12:fault
2223 name="ConnectivityTestError" use="literal"/> </wsdl:fault>
2224 </wsdl:operation>
2225 <wsdl:operation name="CheckMessageStatus">
2226 <soap12:operation soapAction="http://mades.entsoe.eu/CheckMessageStatus"/>
2227 <wsdl:input> <soap12:body use="literal"/> </wsdl:input>
2228 <wsdl:output> <soap12:body use="literal"/> </wsdl:output>
2229 <wsdl:fault name="CheckMessageStatusError"> <soap12:fault
2230 name="CheckMessageStatusError" use="literal"/> </wsdl:fault>
2231 </wsdl:operation>
2232 </wsdl:binding>
2233
2234 <wsdl:binding name="MadesEndpointSOAP11" type="mades:MadesEndpoint">
2235 <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
2236 <wsdl:operation name="SendMessage">
2237 <soap:operation soapAction="http://mades.entsoe.eu/SendMessage"/>
2238 <wsdl:input> <soap:body use="literal"/> </wsdl:input>
2239 <wsdl:output> <soap:body use="literal"/> </wsdl:output>
2240 <wsdl:fault name="SendMessageError"> <soap:fault name="SendMessageError"
2241 use="literal"/> </wsdl:fault>
2242 </wsdl:operation>
2243 <wsdl:operation name="ReceiveMessage">
2244 <soap:operation soapAction="http://mades.entsoe.eu/ReceiveMessage"/>
2245 <wsdl:input> <soap:body use="literal"/> </wsdl:input>
2246 <wsdl:output> <soap:body use="literal"/> </wsdl:output>
2247 <wsdl:fault name="ReceiveMessageError"> <soap:fault name="ReceiveMessageError"
2248 use="literal"/> </wsdl:fault>
2249 </wsdl:operation>
2250 <wsdl:operation name="ConfirmReceiveMessage">
2251 <soap:operation soapAction="http://mades.entsoe.eu/ConfirmReceiveMessage"/>
2252 <wsdl:input> <soap:body use="literal"/> </wsdl:input>
2253 <wsdl:output> <soap:body use="literal"/> </wsdl:output>
2254 <wsdl:fault name="ConfirmReceiveMessageError"> <soap:fault
2255 name="ConfirmReceiveMessageError" use="literal"/> </wsdl:fault>
2256 </wsdl:operation>
2257 <wsdl:operation name="ConnectivityTest">
2258 <soap:operation soapAction="http://mades.entsoe.eu/ConnectivityTest"/>
2259 <wsdl:input> <soap:body use="literal"/> </wsdl:input>
2260 <wsdl:output> <soap:body use="literal"/> </wsdl:output>
2261 <wsdl:fault name="ConnectivityTestError"> <soap:fault
2262 name="ConnectivityTestError" use="literal"/> </wsdl:fault>
2263 </wsdl:operation>
2264 <wsdl:operation name="CheckMessageStatus">
2265 <soap:operation soapAction="http://mades.entsoe.eu/CheckMessageStatus"/>
2266 <wsdl:input> <soap:body use="literal"/> </wsdl:input>
2267 <wsdl:output> <soap:body use="literal"/> </wsdl:output>
2268 <wsdl:fault name="CheckMessageStatusError"> <soap:fault
2269 name="CheckMessageStatusError" use="literal"/> </wsdl:fault>
2270 </wsdl:operation>
2271 </wsdl:binding>
2272
2273 <wsdl:service name="MadesEndpointService">
2274 <wsdl:port name="MadesEndpointSOAP12" binding="mades:MadesEndpointSOAP12">
2275 <soap12:address location="http://mades.entsoe.eu"/>
2276 </wsdl:port>
2277 <wsdl:port name="MadesEndpointSOAP11" binding="mades:MadesEndpointSOAP11">
2278 <soap:address location="http://mades.entsoe.eu"/>
2279 </wsdl:port>
2280 </wsdl:service>
2281 </wsdl:definitions>

```

2282 5.6.3 Node interface

2283 5.6.3.1 Authentication service

```

2284 <?xml version="1.0" encoding="UTF-8"?>
2285 <wsdl:definitions name="MadesAuthenticationService"
2286 targetNamespace="http://mades.entsoe.eu/"
2287 xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
2288 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
2289 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
2290 xmlns:ecp="http://mades.entsoe.eu/"
2291 xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/">
2292
2293 <wsdl:types>
2294 <xsd:schema targetNamespace="http://mades.entsoe.eu/">
2295
2296 <xsd:element name="GetAuthenticationTokenRequest">
2297 <xsd:complexType>
2298 <xsd:sequence>
2299 <xsd:element name="componentCode" type="xsd:string"/>
2300 </xsd:sequence>
2301 </xsd:complexType>
2302 </xsd:element>
2303
2304 <xsd:element name="GetAuthenticationTokenResponse">
2305 <xsd:complexType>
2306 <xsd:sequence>
2307 <xsd:element name="authToken" type="xsd:string"/>
2308 <xsd:element name="expiration" type="xsd:long"/>
2309 <xsd:element minOccurs="0" name="serviceMversion" nillable="true"
2310 type="xsd:int"/>
2311 </xsd:sequence>
2312 </xsd:complexType>
2313 </xsd:element>
2314
2315 <xsd:element name="GetAuthenticationTokenError">
2316 <xsd:complexType>
2317 <xsd:sequence>
2318 <xsd:element name="errorCode" type="xsd:string"/>
2319 <xsd:element name="errorID" type="xsd:string"/>
2320 <xsd:element name="errorMessage" type="xsd:string"/>
2321 <xsd:element minOccurs="0" name="errorDetails" type="xsd:string"/>
2322 </xsd:sequence>
2323 </xsd:complexType>
2324 </xsd:element>
2325 </xsd:schema>
2326 </wsdl:types>
2327
2328 <wsdl:message name="GetAuthenticationTokenResponse">
2329 <wsdl:part name="parameters" element="mades:GetAuthenticationTokenResponse"/>
2330 </wsdl:message>
2331
2332 <wsdl:message name="GetAuthenticationTokenFault">
2333 <wsdl:part name="fault" element="mades:GetAuthenticationTokenError"/>
2334 </wsdl:message>
2335
2336 <wsdl:message name="GetAuthenticationTokenRequest">
2337 <wsdl:part name="parameters" element="mades:GetAuthenticationTokenRequest"/>
2338 </wsdl:message>
2339
2340 <wsdl:portType name="MadesAuthenticationService">
2341 <wsdl:operation name="GetAuthenticationToken">
2342 <wsdl:input message="mades:GetAuthenticationTokenRequest"/>
2343 <wsdl:output message="mades:GetAuthenticationTokenResponse"/>
2344 <wsdl:fault name="GetAuthenticationTokenError"
2345 message="mades:GetAuthenticationTokenFault"/>
2346 </wsdl:operation>

```

```

2347     </wsdl:portType>
2348
2349     <wsdl:binding name="MadesAuthenticationServiceSOAP12"
2350 type="mades:MadesAuthenticationService">
2351     <soap12:binding style="document"
2352 transport="http://schemas.xmlsoap.org/soap/http"/>
2353     <wsdl:operation name="GetAuthenticationToken">
2354     <soap12:operation soapAction="http://mades.entsoe.eu/GetAuthenticationToken"/>
2355     <wsdl:input> <soap12:body use="literal"/> </wsdl:input>
2356     <wsdl:output> <soap12:body use="literal"/> </wsdl:output>
2357     <wsdl:fault name="GetAuthenticationTokenError"> <soap12:fault
2358 name="GetAuthenticationTokenError" use="literal"/> </wsdl:fault>
2359     </wsdl:operation>
2360 </wsdl:binding>
2361
2362 <wsdl:binding name="MadesAuthenticationServiceSOAP11"
2363 type="mades:MadesAuthenticationService">
2364 <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
2365 <wsdl:operation name="GetAuthenticationToken">
2366 <soap:operation soapAction="http://mades.entsoe.eu/GetAuthenticationToken"/>
2367 <wsdl:input> <soap:body use="literal"/> </wsdl:input>
2368 <wsdl:output> <soap:body use="literal"/> </wsdl:output>
2369 <wsdl:fault name="GetAuthenticationTokenError"> <soap:fault
2370 name="GetAuthenticationTokenError" use="literal"/> </wsdl:fault>
2371 </wsdl:operation>
2372 </wsdl:binding>
2373
2374 <wsdl:service name="MadesAuthenticationService">
2375 <wsdl:port name="MadesAuthenticationServiceSOAP12"
2376 binding="mades:MadesAuthenticationServiceSOAP12">
2377 <soap12:address location="http://mades.entsoe.eu"/>
2378 </wsdl:port>
2379 <wsdl:port name="MadesAuthenticationServiceSOAP11"
2380 binding="mades:MadesAuthenticationServiceSOAP11">
2381 <soap:address location="http://mades.entsoe.eu"/>
2382 </wsdl:port>
2383 </wsdl:service>
2384 </wsdl:definitions>

```

2385 5.6.3.2 Messaging services

```

2386 <?xml version="1.0" encoding="UTF-8"?>
2387 <wsdl:definitions name="MadesInternalMessaging"
2388 targetNamespace="http://mades.entsoe.eu/"
2389 xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
2390 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
2391 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
2392 xmlns:ecp="http://mades.entsoe.eu/"
2393 xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/">
2394
2395 <wsdl:types>
2396 <xsd:schema targetNamespace="http://mades.entsoe.eu/">
2397
2398 <xsd:element name="UploadMessagesRequest">
2399 <xsd:complexType>
2400 <xsd:sequence>
2401 <xsd:element maxOccurs="unbounded" name="messages"
2402 type="mades:InternalMessage"/>
2403 <xsd:element name="authToken" type="mades:AuthenticationToken"/>
2404 <xsd:element minOccurs="0" name="serviceMversion" nillable="true"
2405 type="xsd:int"/>
2406 </xsd:sequence>
2407 </xsd:complexType>
2408 </xsd:element>
2409
2410 <xsd:element name="UploadMessagesResponse">
2411 <xsd:complexType>

```



```

2412         <xsd:sequence>
2413             <xsd:element maxOccurs="unbounded" minOccurs="0" name="uploadedMessages"
2414 type="xsd:string"/>
2415             <xsd:element maxOccurs="unbounded" minOccurs="0"
2416 name="notUploadedMessages" type="mades:NotUploadedMessageResponse"/>
2417         </xsd:sequence>
2418     </xsd:complexType>
2419 </xsd:element>
2420
2421 <xsd:element name="UploadMessagesError">
2422     <xsd:complexType>
2423         <xsd:sequence>
2424             <xsd:element name="errorCode" type="xsd:string"/>
2425             <xsd:element name="errorID" type="xsd:string"/>
2426             <xsd:element name="errorMessage" type="xsd:string"/>
2427             <xsd:element minOccurs="0" name="errorDetails" type="xsd:string"/>
2428         </xsd:sequence>
2429     </xsd:complexType>
2430 </xsd:element>
2431
2432 <xsd:element name="DownloadMessagesRequest">
2433     <xsd:complexType>
2434         <xsd:sequence>
2435             <xsd:element maxOccurs="unbounded" name="endpoints"
2436 type="mades:Endpoint"/>
2437             <xsd:element name="authToken" type="mades:AuthenticationToken"/>
2438             <xsd:element minOccurs="0" name="serviceMversion" nillable="true"
2439 type="xsd:int"/>
2440         </xsd:sequence>
2441     </xsd:complexType>
2442 </xsd:element>
2443
2444 <xsd:element name="DownloadMessagesResponse">
2445     <xsd:complexType>
2446         <xsd:sequence>
2447             <xsd:element maxOccurs="unbounded" minOccurs="0" name="messages"
2448 type="mades:InternalMessage"/>
2449             <xsd:element name="waitingMessages" type="xsd:int"/>
2450         </xsd:sequence>
2451     </xsd:complexType>
2452 </xsd:element>
2453
2454 <xsd:element name="DownloadMessagesError">
2455     <xsd:complexType>
2456         <xsd:sequence>
2457             <xsd:element name="errorCode" type="xsd:string"/>
2458             <xsd:element name="errorID" type="xsd:string"/>
2459             <xsd:element name="errorMessage" type="xsd:string"/>
2460             <xsd:element minOccurs="0" name="errorDetails" type="xsd:string"/>
2461         </xsd:sequence>
2462     </xsd:complexType>
2463 </xsd:element>
2464
2465 <xsd:element name="ConfirmDownloadRequest">
2466     <xsd:complexType>
2467         <xsd:sequence>
2468             <xsd:element maxOccurs="unbounded" minOccurs="0" name="messageIDs"
2469 type="xsd:string"/>
2470             <xsd:element name="authToken" type="mades:AuthenticationToken"/>
2471             <xsd:element minOccurs="0" name="serviceMversion" nillable="true"
2472 type="xsd:int"/>
2473         </xsd:sequence>
2474     </xsd:complexType>
2475 </xsd:element>
2476
2477 <xsd:element name="ConfirmDownloadResponse">
2478     <xsd:complexType>

```



```

2479         <xsd:sequence>
2480             <xsd:element maxOccurs="unbounded" minOccurs="0" name="confirmedMessages"
2481 type="xsd:string"/>
2482             <xsd:element maxOccurs="unbounded" minOccurs="0"
2483 name="notConfirmedMessages" type="mades:NotConfirmedMessageResponse"/>
2484         </xsd:sequence>
2485     </xsd:complexType>
2486 </xsd:element>
2487
2488     <xsd:element name="ConfirmDownloadError">
2489         <xsd:complexType>
2490             <xsd:sequence>
2491                 <xsd:element name="errorCode" type="xsd:string"/>
2492                 <xsd:element name="errorID" type="xsd:string"/>
2493                 <xsd:element name="errorMessage" type="xsd:string"/>
2494                 <xsd:element minOccurs="0" name="errorDetails" type="xsd:string"/>
2495             </xsd:sequence>
2496         </xsd:complexType>
2497     </xsd:element>
2498
2499     <xsd:complexType name="InternalMessage">
2500         <xsd:sequence>
2501             <xsd:element name="messageID" type="xsd:string"/>
2502             <xsd:element name="receiverCode" type="xsd:string"/>
2503             <xsd:element name="businessType" type="xsd:string"/>
2504             <xsd:element name="content" type="xsd:base64Binary"/>
2505             <xsd:element minOccurs="0" name="extension" nillable="true"
2506 type="xsd:string"/>
2507             <xsd:element name="generated" type="xsd:dateTime"/>
2508             <xsd:element minOccurs="0" name="expirationTime" nillable="true"
2509 type="xsd:long"/>
2510             <xsd:element name="senderCode" type="xsd:string"/>
2511             <xsd:element name="senderDescription" type="xsd:string"/>
2512             <xsd:element name="internalType" type="mades:InternalMessageType"/>
2513             <xsd:element minOccurs="0" name="relatedMessageID" nillable="true"
2514 type="xsd:string"/>
2515             <xsd:element minOccurs="0" name="senderApplication" nillable="true"
2516 type="xsd:string"/>
2517             <xsd:element minOccurs="0" name="baMessageID" nillable="true"
2518 type="xsd:string"/>
2519             <xsd:element name="metadata" type="mades:MessageMetadata"/>
2520             <xsd:element minOccurs="0" name="messageMversion" nillable="true"
2521 type="xsd:int"/>
2522         </xsd:sequence>
2523     </xsd:complexType>
2524
2525     <xsd:simpleType name="InternalMessageType">
2526         <xsd:restriction base="xsd:string">
2527             <xsd:enumeration value="STANDARD_MESSAGE"/>
2528             <xsd:enumeration value="DELIVERY_ACKNOWLEDGEMENT"/>
2529             <xsd:enumeration value="RECEIVE_ACKNOWLEDGEMENT"/>
2530             <xsd:enumeration value="FAILURE_ACKNOWLEDGEMENT"/>
2531             <xsd:enumeration value="TRACING_MESSAGE"/>
2532             <xsd:enumeration value="TRACING_ACKNOWLEDGEMENT"/>
2533         </xsd:restriction>
2534     </xsd:simpleType>
2535
2536     <xsd:complexType name="MessageMetadata">
2537         <xsd:sequence>
2538             <xsd:element maxOccurs="unbounded" minOccurs="0" name="messageProcessors"
2539 type="mades:MessageProcessor"/>
2540         </xsd:sequence>
2541     </xsd:complexType>
2542
2543     <xsd:complexType name="MessageProcessor">
2544         <xsd:sequence>
2545             <xsd:element name="processorID" type="xsd:string"/>

```

```

2546         <xsd:element name="processorData" type="mades:Map"/>
2547     </xsd:sequence>
2548 </xsd:complexType>
2549
2550 <xsd:complexType name="NotUploadedMessageResponse">
2551     <xsd:sequence>
2552         <xsd:element name="messageID" type="xsd:string"/>
2553         <xsd:element name="fatal" type="xsd:boolean"/>
2554         <xsd:element minOccurs="0" name="businessErrorMessage" nillable="true"
2555 type="xsd:string"/>
2556         <xsd:element name="errorCode" type="xsd:string"/>
2557         <xsd:element name="errorID" type="xsd:string"/>
2558         <xsd:element name="errorMessage" type="xsd:string"/>
2559         <xsd:element minOccurs="0" name="errorDetails" type="xsd:string"/>
2560     </xsd:sequence>
2561 </xsd:complexType>
2562
2563 <xsd:complexType name="NotConfirmedMessageResponse">
2564     <xsd:sequence>
2565         <xsd:element name="messageID" type="xsd:string"/>
2566         <xsd:element name="errorCode" type="xsd:string"/>
2567         <xsd:element name="errorID" type="xsd:string"/>
2568         <xsd:element name="errorMessage" type="xsd:string"/>
2569         <xsd:element minOccurs="0" name="errorDetails" type="xsd:string"/>
2570     </xsd:sequence>
2571 </xsd:complexType>
2572
2573 <xsd:complexType name="Endpoint">
2574     <xsd:sequence>
2575         <xsd:element name="code" type="xsd:string"/>
2576         <xsd:element name="signature" type="xsd:string"/>
2577         <xsd:element name="certificateID" type="xsd:string"/>
2578     </xsd:sequence>
2579 </xsd:complexType>
2580
2581 <xsd:complexType name="AuthenticationToken">
2582     <xsd:sequence>
2583         <xsd:element name="token" type="xsd:string"/>
2584         <xsd:element name="signature" type="xsd:string"/>
2585         <xsd:element name="certificateID" type="xsd:string"/>
2586     </xsd:sequence>
2587 </xsd:complexType>
2588
2589 <xsd:complexType name="Map">
2590     <xsd:sequence>
2591         <xsd:element maxOccurs="unbounded" minOccurs="0" name="entries"
2592 type="mades:MapEntry"/>
2593     </xsd:sequence>
2594 </xsd:complexType>
2595
2596 <xsd:complexType name="MapEntry">
2597     <xsd:sequence>
2598         <xsd:element name="key" type="xsd:string"/>
2599         <xsd:element name="type" type="mades:ValueType"/>
2600         <xsd:element name="value" type="xsd:string"/>
2601     </xsd:sequence>
2602 </xsd:complexType>
2603
2604 <xsd:simpleType name="ValueType">
2605     <xsd:restriction base="xsd:string">
2606         <xsd:enumeration value="STRING"/>
2607         <xsd:enumeration value="LONG"/>
2608         <xsd:enumeration value="BYTE_ARRAY"/>
2609         <xsd:enumeration value="BOOLEAN"/>
2610     </xsd:restriction>
2611 </xsd:simpleType>
2612 </xsd:schema>

```

```

2613     </wsdl:types>
2614
2615     <wsdl:message name="ConfirmDownloadResponse">
2616       <wsdl:part name="parameters" element="mades:ConfirmDownloadResponse"/>
2617     </wsdl:message>
2618
2619     <wsdl:message name="UploadMessagesFault">
2620       <wsdl:part name="fault" element="mades:UploadMessagesError"/>
2621     </wsdl:message>
2622
2623     <wsdl:message name="UploadMessagesRequest">
2624       <wsdl:part name="parameters" element="mades:UploadMessagesRequest"/>
2625     </wsdl:message>
2626
2627     <wsdl:message name="DownloadMessagesFault">
2628       <wsdl:part name="fault" element="mades:DownloadMessagesError"/>
2629     </wsdl:message>
2630
2631     <wsdl:message name="UploadMessagesResponse">
2632       <wsdl:part name="parameters" element="mades:UploadMessagesResponse"/>
2633     </wsdl:message>
2634
2635     <wsdl:message name="DownloadMessagesRequest">
2636       <wsdl:part name="parameters" element="mades:DownloadMessagesRequest"/>
2637     </wsdl:message>
2638
2639     <wsdl:message name="ConfirmDownloadFault">
2640       <wsdl:part name="fault" element="mades:ConfirmDownloadError"/>
2641     </wsdl:message>
2642
2643     <wsdl:message name="DownloadMessagesResponse">
2644       <wsdl:part name="parameters" element="mades:DownloadMessagesResponse"/>
2645     </wsdl:message>
2646
2647     <wsdl:message name="ConfirmDownloadRequest">
2648       <wsdl:part name="parameters" element="mades:ConfirmDownloadRequest"/>
2649     </wsdl:message>
2650
2651     <wsdl:portType name="MadesInternalMessaging">
2652       <wsdl:operation name="UploadMessages">
2653         <wsdl:input message="mades:UploadMessagesRequest"/>
2654         <wsdl:output message="mades:UploadMessagesResponse"/>
2655         <wsdl:fault name="UploadMessagesError" message="mades:UploadMessagesFault"/>
2656       </wsdl:operation>
2657       <wsdl:operation name="DownloadMessages">
2658         <wsdl:input message="mades:DownloadMessagesRequest"/>
2659         <wsdl:output message="mades:DownloadMessagesResponse"/>
2660         <wsdl:fault name="DownloadMessagesError"
2661 message="mades:DownloadMessagesFault"/>
2662       </wsdl:operation>
2663       <wsdl:operation name="ConfirmDownload">
2664         <wsdl:input message="mades:ConfirmDownloadRequest"/>
2665         <wsdl:output message="mades:ConfirmDownloadResponse"/>
2666         <wsdl:fault name="ConfirmDownloadError" message="mades:ConfirmDownloadFault"/>
2667       </wsdl:operation>
2668     </wsdl:portType>
2669
2670     <wsdl:binding name="MadesInternalMessagingSOAP11"
2671 type="mades:MadesInternalMessaging">
2672       <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
2673       <wsdl:operation name="UploadMessages">
2674         <soap:operation soapAction="http://mades.entsoe.eu/UploadMessages"/>
2675         <wsdl:input> <soap:body use="literal"/> </wsdl:input>
2676         <wsdl:output> <soap:body use="literal"/> </wsdl:output>
2677         <wsdl:fault name="UploadMessagesError"> <soap:fault name="UploadMessagesError"
2678 use="literal"/> </wsdl:fault>
2679       </wsdl:operation>

```

```

2680     <wsdl:operation name="DownloadMessages">
2681         <soap:operation soapAction="http://mades.entsoe.eu/DownloadMessages"/>
2682         <wsdl:input> <soap:body use="literal"/> </wsdl:input>
2683         <wsdl:output> <soap:body use="literal"/> </wsdl:output>
2684         <wsdl:fault name="DownloadMessagesError"> <soap:fault
2685 name="DownloadMessagesError" use="literal"/> </wsdl:fault>
2686     </wsdl:operation>
2687     <wsdl:operation name="ConfirmDownload">
2688         <soap:operation soapAction="http://mades.entsoe.eu/ConfirmDownload"/>
2689         <wsdl:input> <soap:body use="literal"/> </wsdl:input>
2690         <wsdl:output> <soap:body use="literal"/> </wsdl:output>
2691         <wsdl:fault name="ConfirmDownloadError"> <soap:fault
2692 name="ConfirmDownloadError" use="literal"/> </wsdl:fault>
2693     </wsdl:operation>
2694 </wsdl:binding>
2695
2696     <wsdl:binding name="MadesInternalMessagingSOAP12"
2697 type="mades:MadesInternalMessaging">
2698         <soap12:binding style="document"
2699 transport="http://schemas.xmlsoap.org/soap/http"/>
2700         <wsdl:operation name="UploadMessages">
2701             <soap12:operation soapAction="http://mades.entsoe.eu/UploadMessages"/>
2702             <wsdl:input> <soap12:body use="literal"/> </wsdl:input>
2703             <wsdl:output> <soap12:body use="literal"/> </wsdl:output>
2704             <wsdl:fault name="UploadMessagesError"> <soap12:fault
2705 name="UploadMessagesError" use="literal"/> </wsdl:fault>
2706         </wsdl:operation>
2707         <wsdl:operation name="DownloadMessages">
2708             <soap12:operation soapAction="http://mades.entsoe.eu/DownloadMessages"/>
2709             <wsdl:input> <soap12:body use="literal"/> </wsdl:input>
2710             <wsdl:output> <soap12:body use="literal"/> </wsdl:output>
2711             <wsdl:fault name="DownloadMessagesError"> <soap12:fault
2712 name="DownloadMessagesError" use="literal"/> </wsdl:fault>
2713         </wsdl:operation>
2714         <wsdl:operation name="ConfirmDownload">
2715             <soap12:operation soapAction="http://mades.entsoe.eu/ConfirmDownload"/>
2716             <wsdl:input> <soap12:body use="literal"/> </wsdl:input>
2717             <wsdl:output> <soap12:body use="literal"/> </wsdl:output>
2718             <wsdl:fault name="ConfirmDownloadError"> <soap12:fault
2719 name="ConfirmDownloadError" use="literal"/> </wsdl:fault>
2720         </wsdl:operation>
2721     </wsdl:binding>
2722
2723     <wsdl:service name="MadesInternalMessagingService">
2724         <wsdl:port name="MadesInternalMessagingSOAP11"
2725 binding="mades:MadesInternalMessagingSOAP11">
2726             <soap:address location="http://mades.entsoe.eu"/>
2727         </wsdl:port>
2728         <wsdl:port name="MadesInternalMessagingSOAP12"
2729 binding="mades:MadesInternalMessagingSOAP12">
2730             <soap12:address location="http://mades.entsoe.eu"/>
2731         </wsdl:port>
2732     </wsdl:service>
2733 </wsdl:definitions>

```

2734 5.6.3.3 Directory services

```

2735 <?xml version="1.0" encoding="UTF-8"?>
2736 <wsdl:definitions name="MadesDirectoryService"
2737 targetNamespace="http://mades.entsoe.eu/"
2738 xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
2739 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
2740 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
2741 xmlns:ecp="http://mades.entsoe.eu/"
2742 xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
2743
2744 <wsdl:types>

```

```

2745     <xsd:schema targetNamespace="http://mades.entsoe.eu/">
2746
2747         <xsd:element name="GetCertificateRequest">
2748             <xsd:complexType>
2749                 <xsd:sequence>
2750                     <xsd:element name="componentCode" type="xsd:string"/>
2751                     <xsd:element name="type" type="mades:CertificateType"/>
2752                     <xsd:element minOccurs="0" name="certificateID" nillable="true"
2753 type="xsd:string"/>
2754                     <xsd:element name="authToken" type="mades:AuthenticationToken"/>
2755                     <xsd:element minOccurs="0" name="serviceMversion" nillable="true"
2756 type="xsd:int"/>
2757                 </xsd:sequence>
2758             </xsd:complexType>
2759         </xsd:element>
2760
2761         <xsd:element name="GetCertificateResponse">
2762             <xsd:complexType>
2763                 <xsd:sequence>
2764                     <xsd:element minOccurs="0" name="certificate" nillable="true"
2765 type="mades:Certificate"/>
2766                 </xsd:sequence>
2767             </xsd:complexType>
2768         </xsd:element>
2769
2770         <xsd:element name="GetCertificateError">
2771             <xsd:complexType>
2772                 <xsd:sequence>
2773                     <xsd:element name="errorCode" type="xsd:string"/>
2774                     <xsd:element name="errorID" type="xsd:string"/>
2775                     <xsd:element name="errorMessage" type="xsd:string"/>
2776                     <xsd:element minOccurs="0" name="errorDetails" type="xsd:string"/>
2777                 </xsd:sequence>
2778             </xsd:complexType>
2779         </xsd:element>
2780
2781         <xsd:element name="GetComponentRequest">
2782             <xsd:complexType>
2783                 <xsd:sequence>
2784                     <xsd:element name="componentCode" type="xsd:string"/>
2785                     <xsd:element name="authToken" type="mades:AuthenticationToken"/>
2786                     <xsd:element minOccurs="0" name="serviceMversion" nillable="true"
2787 type="xsd:int"/>
2788                 </xsd:sequence>
2789             </xsd:complexType>
2790         </xsd:element>
2791
2792         <xsd:element name="GetComponentResponse">
2793             <xsd:complexType>
2794                 <xsd:sequence>
2795                     <xsd:element minOccurs="0" name="component" nillable="true"
2796 type="mades:ComponentInformation"/>
2797                 </xsd:sequence>
2798             </xsd:complexType>
2799         </xsd:element>
2800
2801         <xsd:element name="GetComponentError">
2802             <xsd:complexType>
2803                 <xsd:sequence>
2804                     <xsd:element name="errorCode" type="xsd:string"/>
2805                     <xsd:element name="errorID" type="xsd:string"/>
2806                     <xsd:element name="errorMessage" type="xsd:string"/>
2807                     <xsd:element minOccurs="0" name="errorDetails" type="xsd:string"/>
2808                 </xsd:sequence>
2809             </xsd:complexType>
2810         </xsd:element>
2811

```

```

2812     <xsd:element name="SetComponentMversionRequest">
2813         <xsd:complexType>
2814             <xsd:sequence>
2815                 <xsd:element name="componentCode" type="xsd:string"/>
2816                 <xsd:element name="signature" type="xsd:string"/>
2817                 <xsd:element name="certificateID" type="xsd:string"/>
2818                 <xsd:element name="componentMVersion" type="xsd:int"/>
2819                 <xsd:element name="authToken" type="makes:AuthenticationToken"/>
2820                 <xsd:element minOccurs="0" name="serviceMversion" nillable="true"
2821 type="xsd:int"/>
2822             </xsd:sequence>
2823         </xsd:complexType>
2824     </xsd:element>
2825
2826     <xsd:element name="SetComponentMversionResponse">
2827         <xsd:complexType>
2828             <xsd:sequence>
2829                 <xsd:element name="nodeMversion" type="xsd:int"/>
2830                 <xsd:element name="acceptance" type="xsd:boolean"/>
2831             </xsd:sequence>
2832         </xsd:complexType>
2833     </xsd:element>
2834
2835     <xsd:element name="SetComponentMversionError">
2836         <xsd:complexType>
2837             <xsd:sequence>
2838                 <xsd:element name="errorCode" type="xsd:string"/>
2839                 <xsd:element name="errorID" type="xsd:string"/>
2840                 <xsd:element name="errorMessage" type="xsd:string"/>
2841                 <xsd:element minOccurs="0" name="errorDetails" type="xsd:string"/>
2842             </xsd:sequence>
2843         </xsd:complexType>
2844     </xsd:element>
2845
2846     <xsd:complexType name="Certificate">
2847         <xsd:sequence>
2848             <xsd:element name="certificateID" type="xsd:string"/>
2849             <xsd:element name="certificate" type="xsd:base64Binary"/>
2850             <xsd:element name="expiration" type="xsd:long"/>
2851         </xsd:sequence>
2852     </xsd:complexType>
2853
2854     <xsd:simpleType name="CertificateType">
2855         <xsd:restriction base="xsd:string">
2856             <xsd:enumeration value="AUTHENTICATION"/>
2857             <xsd:enumeration value="ENCRYPTION"/>
2858             <xsd:enumeration value="SIGNING"/>
2859         </xsd:restriction>
2860     </xsd:simpleType>
2861
2862     <xsd:complexType name="ComponentInformation">
2863         <xsd:sequence>
2864             <xsd:element name="code" type="xsd:string"/>
2865             <xsd:element name="type" type="makes:ComponentType"/>
2866             <xsd:element name="organization" type="xsd:string"/>
2867             <xsd:element name="person" type="xsd:string"/>
2868             <xsd:element name="email" type="xsd:string"/>
2869             <xsd:element name="phone" type="xsd:string"/>
2870             <xsd:element name="routing" type="makes:RoutingInformation"/>
2871             <xsd:element minOccurs="0" name="expiration" nillable="true"
2872 type="xsd:long"/>
2873             <xsd:element minOccurs="0" name="codeMversion" nillable="true"
2874 type="xsd:int"/>
2875         </xsd:sequence>
2876     </xsd:complexType>
2877
2878     <xsd:complexType name="RoutingInformation">

```



```

2879         <xsd:sequence>
2880             <xsd:element name="node" type="xsd:string"/>
2881             <xsd:element name="primaryURL" type="xsd:string"/>
2882             <xsd:element minOccurs="0" name="secondaryURL" nillable="true"
2883 type="xsd:string"/>
2884             <xsd:element minOccurs="0" name="nodeMversion" nillable="true"
2885 type="xsd:int"/>
2886         </xsd:sequence>
2887     </xsd:complexType>
2888
2889     <xsd:simpleType name="ComponentType">
2890         <xsd:restriction base="xsd:string">
2891             <xsd:enumeration value="NODE"/>
2892             <xsd:enumeration value="ENDPOINT"/>
2893         </xsd:restriction>
2894     </xsd:simpleType>
2895
2896     <xsd:complexType name="AuthenticationToken">
2897         <xsd:sequence>
2898             <xsd:element name="token" type="xsd:string"/>
2899             <xsd:element name="signature" type="xsd:string"/>
2900             <xsd:element name="certificateID" type="xsd:string"/>
2901         </xsd:sequence>
2902     </xsd:complexType>
2903 </xsd:schema>
2904 </wsdl:types>
2905
2906 <wsdl:message name="GetComponentRequest">
2907     <wsdl:part name="parameters" element="mades:GetComponentRequest"/>
2908 </wsdl:message>
2909
2910 <wsdl:message name="GetCertificateResponse">
2911     <wsdl:part name="parameters" element="mades:GetCertificateResponse"/>
2912 </wsdl:message>
2913
2914 <wsdl:message name="GetComponentResponse">
2915     <wsdl:part name="parameters" element="mades:GetComponentResponse"/>
2916 </wsdl:message>
2917
2918 <wsdl:message name="GetCertificateRequest">
2919     <wsdl:part name="parameters" element="mades:GetCertificateRequest"/>
2920 </wsdl:message>
2921
2922 <wsdl:message name="GetCertificateFault">
2923     <wsdl:part name="fault" element="mades:GetCertificateError"/>
2924 </wsdl:message>
2925
2926 <wsdl:message name="GetComponentFault">
2927     <wsdl:part name="fault" element="mades:GetComponentError"/>
2928 </wsdl:message>
2929
2930 <wsdl:message name="SetComponentMversionRequest">
2931     <wsdl:part name="parameters" element="mades:SetComponentMversionRequest"/>
2932 </wsdl:message>
2933
2934 <wsdl:message name="SetComponentMversionResponse">
2935     <wsdl:part name="parameters" element="mades:SetComponentMversionResponse"/>
2936 </wsdl:message>
2937
2938 <wsdl:message name="SetComponentMversionFault">
2939     <wsdl:part name="fault" element="mades:SetComponentMversionError"/>
2940 </wsdl:message>
2941
2942 <wsdl:portType name="MadesDirectoryService">
2943     <wsdl:operation name="GetCertificate">
2944         <wsdl:input message="mades:GetCertificateRequest"/>
2945         <wsdl:output message="mades:GetCertificateResponse"/>

```



```

2946     <wsdl:fault name="GetCertificateError" message="mades:GetCertificateFault"/>
2947 </wsdl:operation>
2948 <wsdl:operation name="GetComponent">
2949     <wsdl:input message="mades:GetComponentRequest"/>
2950     <wsdl:output message="mades:GetComponentResponse"/>
2951     <wsdl:fault name="GetComponentError" message="mades:GetComponentFault"/>
2952 </wsdl:operation>
2953 <wsdl:operation name="SetComponentMversion">
2954     <wsdl:input message="mades:SetComponentMversionRequest"/>
2955     <wsdl:output message="mades:SetComponentMversionResponse"/>
2956     <wsdl:fault name="SetComponentMversionError"
2957 message="mades:SetComponentMversionFault"/>
2958 </wsdl:operation>
2959 </wsdl:portType>
2960
2961 <wsdl:binding name="MadesDirectoryServiceSOAP11"
2962 type="mades:MadesDirectoryService">
2963     <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
2964     <wsdl:operation name="GetCertificate">
2965         <soap:operation soapAction="http://mades.entsoe.eu/GetCertificate"/>
2966         <wsdl:input> <soap:body use="literal"/> </wsdl:input>
2967         <wsdl:output> <soap:body use="literal"/> </wsdl:output>
2968         <wsdl:fault name="GetCertificateError"> <soap:fault name="GetCertificateError"
2969 use="literal"/> </wsdl:fault>
2970     </wsdl:operation>
2971     <wsdl:operation name="GetComponent">
2972         <soap:operation soapAction="http://mades.entsoe.eu/GetComponent"/>
2973         <wsdl:input> <soap:body use="literal"/> </wsdl:input>
2974         <wsdl:output> <soap:body use="literal"/> </wsdl:output>
2975         <wsdl:fault name="GetComponentError"> <soap:fault name="GetComponentError"
2976 use="literal"/> </wsdl:fault>
2977     </wsdl:operation>
2978     <wsdl:operation name="SetComponentMversion">
2979         <soap:operation soapAction="http://mades.entsoe.eu/SetComponentMversion"/>
2980         <wsdl:input> <soap:body use="literal"/> </wsdl:input>
2981         <wsdl:output> <soap:body use="literal"/> </wsdl:output>
2982         <wsdl:fault name="SetComponentMversionError"> <soap:fault
2983 name="SetComponentMversionError" use="literal"/> </wsdl:fault>
2984     </wsdl:operation>
2985 </wsdl:binding>
2986
2987 <wsdl:binding name="MadesDirectoryServiceSOAP12"
2988 type="mades:MadesDirectoryService">
2989     <soap12:binding style="document"
2990 transport="http://schemas.xmlsoap.org/soap/http"/>
2991     <wsdl:operation name="GetCertificate">
2992         <soap12:operation soapAction="http://mades.entsoe.eu/GetCertificate"/>
2993         <wsdl:input> <soap12:body use="literal"/> </wsdl:input>
2994         <wsdl:output> <soap12:body use="literal"/> </wsdl:output>
2995         <wsdl:fault name="GetCertificateError"> <soap12:fault
2996 name="GetCertificateError" use="literal"/> </wsdl:fault>
2997     </wsdl:operation>
2998     <wsdl:operation name="GetComponent">
2999         <soap12:operation soapAction="http://mades.entsoe.eu/GetComponent"/>
3000         <wsdl:input> <soap12:body use="literal"/> </wsdl:input>
3001         <wsdl:output> <soap12:body use="literal"/> </wsdl:output>
3002         <wsdl:fault name="GetComponentError"> <soap12:fault name="GetComponentError"
3003 use="literal"/> </wsdl:fault>
3004     </wsdl:operation>
3005     <wsdl:operation name="SetComponentMversion">
3006         <soap12:operation soapAction="http://mades.entsoe.eu/SetComponentMversion"/>
3007         <wsdl:input> <soap:body use="literal"/> </wsdl:input>
3008         <wsdl:output> <soap:body use="literal"/> </wsdl:output>
3009         <wsdl:fault name="SetComponentMversionError"> <soap12:fault
3010 name="SetComponentMversionError" use="literal"/> </wsdl:fault>
3011     </wsdl:operation>
3012 </wsdl:binding>

```

```

3013
3014     <wsdl:service name="MadesDirectoryService">
3015         <wsdl:port name="MadesDirectoryServiceSOAP11"
3016         binding="mades:MadesDirectoryServiceSOAP11">
3017             <soap:address location="http://mades.entsoe.eu"/>
3018         </wsdl:port>
3019         <wsdl:port name="MadesDirectoryServiceSOAP12"
3020         binding="mades:MadesDirectoryServiceSOAP12">
3021             <soap12:address location="http://mades.entsoe.eu"/>
3022         </wsdl:port>
3023     </wsdl:service>
3024 </wsdl:definitions>

```

3025 5.6.3.4 Node synchronization interface

```

3026 <?xml version="1.0" encoding="UTF-8"?>
3027 <wsdl:definitions name="MadesNodeSynchronizationService"
3028 targetNamespace="http://mades.entsoe.eu/"
3029 xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
3030 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
3031 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
3032 xmlns:ecp="http://mades.entsoe.eu/"
3033 xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/">
3034
3035     <wsdl:types>
3036         <xsd:schema targetNamespace="http://mades.entsoe.eu/">
3037
3038             <xsd:element name="GetAllDirectoryDataRequest">
3039                 <xsd:complexType>
3040                     <xsd:sequence>
3041                         <xsd:element minOccurs="0" name="dversion" nillable="true"
3042                         type="xsd:int"/>
3043                         <xsd:element minOccurs="0" name="serviceMversion" nillable="true"
3044                         type="xsd:int"/>
3045                     </xsd:sequence>
3046                 </xsd:complexType>
3047             </xsd:element>
3048
3049             <xsd:element name="GetAllDirectoryDataResponse">
3050                 <xsd:complexType>
3051                     <xsd:sequence>
3052                         <xsd:element name="dversion" type="xsd:int"/>
3053                         <xsd:element name="nodeCode" type="xsd:string"/>
3054                         <xsd:element minOccurs="unbounded" minOccurs="0" name="components"
3055                         nillable="true" type="mades:ComponentDescription"/>
3056                     </xsd:sequence>
3057                 </xsd:complexType>
3058             </xsd:element>
3059
3060             <xsd:element name="GetAllDirectoryDataError">
3061                 <xsd:complexType>
3062                     <xsd:sequence>
3063                         <xsd:element name="errorCode" type="xsd:string"/>
3064                         <xsd:element name="errorID" type="xsd:string"/>
3065                         <xsd:element name="errorMessage" type="xsd:string"/>
3066                         <xsd:element minOccurs="0" name="errorDetails" type="xsd:string"/>
3067                     </xsd:sequence>
3068                 </xsd:complexType>
3069             </xsd:element>
3070
3071             <xsd:element name="GetNodeMversionRequest">
3072                 <xsd:complexType>
3073                     <xsd:sequence>
3074                         <xsd:element name="mversion" type="xsd:int"/>
3075                     </xsd:sequence>
3076                 </xsd:complexType>
3077             </xsd:element>

```

```

3078
3079     <xsd:element name="GetNodeMversionResponse">
3080         <xsd:complexType>
3081             <xsd:sequence>
3082                 <xsd:element name="mversion" type="xsd:int"/>
3083                 <xsd:element name="nodeCode" type="xsd:string"/>
3084             </xsd:sequence>
3085         </xsd:complexType>
3086     </xsd:element>
3087
3088     <xsd:element name="GetNodeMversionError">
3089         <xsd:complexType>
3090             <xsd:sequence>
3091                 <xsd:element name="errorCode" type="xsd:string"/>
3092                 <xsd:element name="errorID" type="xsd:string"/>
3093                 <xsd:element name="errorMessage" type="xsd:string"/>
3094                 <xsd:element minOccurs="0" name="errorDetails" type="xsd:string"/>
3095             </xsd:sequence>
3096         </xsd:complexType>
3097     </xsd:element>
3098
3099     <xsd:complexType name="ComponentDescription">
3100         <xsd:sequence>
3101             <xsd:element name="information" type="mades:ComponentInformation"/>
3102             <xsd:element maxOccurs="unbounded" minOccurs="0" name="certificates"
3103 type="mades:ComponentCertificate"/>
3104         </xsd:sequence>
3105     </xsd:complexType>
3106
3107     <xsd:complexType name="ComponentCertificate">
3108         <xsd:sequence>
3109             <xsd:element name="certificate" type="mades:Certificate"/>
3110             <xsd:element minOccurs="0" name="revoked" nillable="true"
3111 type="xsd:boolean"/>
3112             <xsd:element name="type" type="mades:CertificateType"/>
3113         </xsd:sequence>
3114     </xsd:complexType>
3115
3116     <xsd:complexType name="Certificate">
3117         <xsd:sequence>
3118             <xsd:element name="certificateID" type="xsd:string"/>
3119             <xsd:element name="certificate" type="xsd:base64Binary"/>
3120         </xsd:sequence>
3121     </xsd:complexType>
3122
3123     <xsd:simpleType name="CertificateType">
3124         <xsd:restriction base="xsd:string">
3125             <xsd:enumeration value="AUTHENTICATION"/>
3126             <xsd:enumeration value="ENCRYPTION"/>
3127             <xsd:enumeration value="SIGNING"/>
3128         </xsd:restriction>
3129     </xsd:simpleType>
3130
3131     <xsd:complexType name="ComponentInformation">
3132         <xsd:sequence>
3133             <xsd:element name="code" type="xsd:string"/>
3134             <xsd:element name="type" type="mades:ComponentType"/>
3135             <xsd:element name="organization" type="xsd:string"/>
3136             <xsd:element name="person" type="xsd:string"/>
3137             <xsd:element name="email" type="xsd:string"/>
3138             <xsd:element name="phone" type="xsd:string"/>
3139             <xsd:element name="routing" type="mades:RoutingInformation"/>
3140             <xsd:element minOccurs="0" name="expiration" nillable="true"
3141 type="xsd:long"/>
3142             <xsd:element minOccurs="0" name="codeMversion" nillable="true"
3143 type="xsd:int"/>
3144         </xsd:sequence>

```

```

3145     </xsd:complexType>
3146
3147     <xsd:complexType name="RoutingInformation">
3148         <xsd:sequence>
3149             <xsd:element name="node" type="xsd:string"/>
3150             <xsd:element name="primaryURL" type="xsd:string"/>
3151             <xsd:element minOccurs="0" name="secondaryURL" nillable="true"
3152 type="xsd:string"/>
3153             <xsd:element minOccurs="0" name="nodeMversion" nillable="true"
3154 type="xsd:int"/>
3155         </xsd:sequence>
3156     </xsd:complexType>
3157
3158     <xsd:simpleType name="ComponentType">
3159         <xsd:restriction base="xsd:string">
3160             <xsd:enumeration value="NODE"/>
3161             <xsd:enumeration value="ENDPOINT"/>
3162         </xsd:restriction>
3163     </xsd:simpleType>
3164
3165 </xsd:schema>
3166 </wsdl:types>
3167
3168 <wsdl:message name="GetAllDirectoryDataResponse">
3169     <wsdl:part name="parameters" element="mades:GetAllDirectoryDataResponse"/>
3170 </wsdl:message>
3171
3172 <wsdl:message name="GetAllDirectoryDataFault">
3173     <wsdl:part name="fault" element="mades:GetAllDirectoryDataError"/>
3174 </wsdl:message>
3175
3176 <wsdl:message name="GetAllDirectoryDataRequest">
3177     <wsdl:part name="parameters" element="mades:GetAllDirectoryDataRequest"/>
3178 </wsdl:message>
3179
3180 <wsdl:message name="GetNodeMversionResponse">
3181     <wsdl:part name="parameters" element="mades:GetNodeMversionResponse"/>
3182 </wsdl:message>
3183
3184 <wsdl:message name="GetNodeMversionFault">
3185     <wsdl:part name="fault" element="mades:GetNodeMversionError"/>
3186 </wsdl:message>
3187
3188 <wsdl:message name="GetNodeMversionRequest">
3189     <wsdl:part name="parameters" element="mades:GetNodeMversionRequest"/>
3190 </wsdl:message>
3191
3192 <wsdl:portType name="MadesNodeSynchronizationService">
3193     <wsdl:operation name="GetAllDirectoryData">
3194         <wsdl:input message="mades:GetAllDirectoryDataRequest"/>
3195         <wsdl:output message="mades:GetAllDirectoryDataResponse"/>
3196         <wsdl:fault name="GetAllDirectoryDataError"
3197 message="mades:GetAllDirectoryDataFault"/>
3198     </wsdl:operation>
3199     <wsdl:operation name="GetNodeMversion">
3200         <wsdl:input message="mades:GetNodeMversionRequest"/>
3201         <wsdl:output message="mades:GetNodeMversionResponse"/>
3202         <wsdl:fault name="GetNodeMversionError" message="mades:GetNodeMversionFault"/>
3203     </wsdl:operation>
3204 </wsdl:portType>
3205
3206 <wsdl:binding name="MadesNodeSynchronizationServiceSOAP12"
3207 type="mades:MadesNodeSynchronizationService">
3208     <soap12:binding style="document"
3209 transport="http://schemas.xmlsoap.org/soap/http"/>
3210     <wsdl:operation name="GetAllDirectoryData">
3211         <soap12:operation soapAction="http://mades.entsoe.eu/GetAllDirectoryData"/>

```

```

3212     <wsdl:input> <soap12:body use="literal"/> </wsdl:input>
3213     <wsdl:output> <soap12:body use="literal"/> </wsdl:output>
3214     <wsdl:fault name="GetAllDirectoryDataError"> <soap12:fault
3215 name="GetAllDirectoryDataError" use="literal"/> </wsdl:fault>
3216 </wsdl:operation>
3217 <wsdl:operation name="GetNodeMversion">
3218   <soap12:operation soapAction="http://mades.entsoe.eu/GetNodeMversion"/>
3219   <wsdl:input> <soap12:body use="literal"/> </wsdl:input>
3220   <wsdl:output> <soap12:body use="literal"/> </wsdl:output>
3221   <wsdl:fault name="GetNodeMversionError"> <soap12:fault
3222 name="GetNodeMversionError" use="literal"/> </wsdl:fault>
3223 </wsdl:operation>
3224 </wsdl:binding>
3225
3226 <wsdl:binding name="MadesNodeSynchronizationServiceSOAP11"
3227 type="mades:MadesNodeSynchronizationService">
3228   <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
3229   <wsdl:operation name="GetAllDirectoryData">
3230     <soap:operation soapAction="http://mades.entsoe.eu/GetAllDirectoryData"/>
3231     <wsdl:input> <soap:body use="literal"/> </wsdl:input>
3232     <wsdl:output> <soap:body use="literal"/> </wsdl:output>
3233     <wsdl:fault name="GetAllDirectoryDataError"> <soap:fault
3234 name="GetAllDirectoryDataError" use="literal"/> </wsdl:fault>
3235   </wsdl:operation>
3236   <wsdl:operation name="GetNodeMversion">
3237     <soap:operation soapAction="http://mades.entsoe.eu/GetNodeMversion"/>
3238     <wsdl:input> <soap:body use="literal"/> </wsdl:input>
3239     <wsdl:output> <soap:body use="literal"/> </wsdl:output>
3240     <wsdl:fault name="GetNodeMversionError"> <soap:fault
3241 name="GetNodeMversionError" use="literal"/> </wsdl:fault>
3242   </wsdl:operation>
3243 </wsdl:binding>
3244
3245 <wsdl:service name="MadesNodeSynchronizationService">
3246   <wsdl:port name="MadesNodeSynchronizationServiceSOAP12"
3247 binding="mades:MadesNodeSynchronizationServiceSOAP12">
3248     <soap12:address location="http://mades.entsoe.eu"/>
3249   </wsdl:port>
3250   <wsdl:port name="MadesNodeSynchronizationServiceSOAP11"
3251 binding="mades:MadesNodeSynchronizationServiceSOAP11">
3252     <soap:address location="http://mades.entsoe.eu"/>
3253   </wsdl:port>
3254 </wsdl:service>
3255 </wsdl:definitions>

```

3256 5.6.4 XML signature example

```

3257 <?xml version="1.0" encoding="UTF-8" standalone="no"?>
3258 <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
3259   <SignedInfo>
3260     <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-
3261 20010315"/>
3262     <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha512"/>
3263     <Reference URI="">
3264       <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha512"/>
3265       <DigestValue>eVpInNsCIWzEjdrxxvongO2rnQ4=</DigestValue>
3266     </Reference>
3267   </SignedInfo>
3268   <SignatureValue>aD9HniTmVxW+HnDopSjzWDB+MypGTC7yb3/HUpAZKmEhRwQC0eBwYcZSRTqF8VdzmneH6abq2P+m
3269 vHNXPc53i3mF58XDR5JFHHWLHq8B9HZm6/IYxcNy2cGW9yAVyQKe3uJXeV/95u9qMEWJhbOjvPIx
3270 ZdbXqcCSorWqih7hdB86Nv2SIBfXmvWdIinwZfU/44RUptNyxQpP/Pw91Dd8YnMTNVwm2ax5oLlW
3271 akIGKToS/yYid/Cgyb1xhGdfXEp30bqLusaLMYkbctpZ2Wdn2w5I4mm0078jndPUnaMT5gyFaonz
3272 +K84xD+1/tZbTQ0adc9LE7XgAkpIinjf2LW9tw==</SignatureValue>
3273   <KeyInfo>
3274     <KeyName>YYYYYYYYYYYYYYYY</KeyName>
3275   </KeyInfo>
3276 </Signature>

```

3277 Where:

- 3278 • DigestValue is the non encoded hash of the message.
- 3279 • SignatureValue is the encoded hash of the message.
- 3280 • KeyName is the ID of the signer component.

3281 _____