

# FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS

## FIPA Abstract Architecture Specification

<b>Document title</b>	FIPA Abstract Architecture Specification		
<b>Document number</b>	XC00001K	<b>Document source</b>	FIPA TC Architecture
<b>Document status</b>	Experimental	<b>Date of this status</b>	2002/ <del>10/1805/0810</del>
<b>Supersedes</b>	None		
<b>Contact</b>	fab@fipa.org		
<b>Change history</b>			
2002/05/ <del>0810</del>	See <del>Informative Annex E — ChangeLog</del> <del>Informative Annex E — ChangeLog</del>		

© ~~1996-2002~~ Foundation for Intelligent Physical Agents

~~http://www.fipa.org/~~

Geneva, Switzerland

Notice

Use of the technologies described in this specification may infringe patents, copyrights or other intellectual property rights of FIPA Members and non-members. Nothing in this specification should be construed as granting permission to use any of the technologies described. Anyone planning to make use of technology covered by the intellectual property rights of others should first obtain permission from the holder(s) of the rights. FIPA strongly encourages anyone implementing any part of this specification to determine first whether part(s) sought to be implemented are covered by the intellectual property of others, and, if so, to obtain appropriate licenses or other permission from the holder(s) of such intellectual property prior to implementation. This specification is subject to change without notice. Neither FIPA nor any of its Members accept any responsibility whatsoever for damages or liability, direct or consequential, which may result from the use of this specification.

## 21 Foreword

22 The Foundation for Intelligent Physical Agents (FIPA) is an international organization that is dedicated to promoting the  
23 industry of intelligent agents by openly developing specifications supporting interoperability among agents and agent-  
24 based applications. This occurs through open collaboration among its member organizations, which are companies  
25 and universities that are active in the field of agents. FIPA makes the results of its activities available to all interested  
26 parties and intends to contribute its results to the appropriate formal standards bodies [where appropriate](#).

27 The members of FIPA are individually and collectively committed to open competition in the development of agent-  
28 based applications, services and equipment. Membership in FIPA is open to any corporation and individual firm,  
29 partnership, governmental body or international organization without restriction. In particular, members are not bound  
30 to implement or use specific agent-based standards, recommendations and FIPA specifications by virtue of their  
31 participation in FIPA.

32 The FIPA specifications are developed through direct involvement of the FIPA membership. The status of a  
33 specification can be Preliminary, Experimental, Standard, Deprecated or Obsolete. More detail about the process of  
34 specification may be found in the FIPA [Document Policy \[f-out-00000\]](#) and the FIPA [Specifications Policy \[f-out-00003\]](#)~~Procedures for Technical Work~~. A complete overview of the FIPA specifications and their current status may be  
35 found ~~in the FIPA List of Specifications. A list of terms and abbreviations used in the FIPA specifications may be found~~  
36 ~~in the FIPA Glossary on the FIPA Web site~~.  
37 [in the FIPA Glossary on the FIPA Web site](#).

38 FIPA is a non-profit association registered in Geneva, Switzerland. As of ~~June~~[January](#) 200~~20~~[29](#), the 56 members of FIPA  
39 represented ~~47~~[many](#) countries worldwide. Further information about FIPA as an organization, membership information,  
40 FIPA specifications and upcoming meetings may be found [on the FIPA Web site](#) at <http://www.fipa.org/>.

41 **Contents**

42	1	Introduction .....	1
43	1.1	Contents .....	1
44	1.2	Audience .....	1
45	1.3	Acknowledgements .....	2
46	2	Scope and Methodology .....	3
47	2.1	Background .....	3
48	2.2	Why an Abstract Architecture? .....	4
49	2.3	Scope of the Abstract Architecture .....	4
50	2.3.1	Areas that are not Sufficiently Abstract.....	5
51	2.3.2	Areas for Future Consideration.....	5
52	2.4	Going From Abstract to Concrete Specifications.....	65
53	2.5	Methodology.....	7
54	2.6	Status of the Abstract Architecture .....	8
55	2.7	Evolution of the Abstract Architecture.....	8
56	3	Themes of the Abstract Architecture .....	9
57	3.1	Focus on Agent Interoperability .....	10
58	3.2	An Exemplar System.....	10
59	4	Architectural Overview .....	11
60	4.1	Agents and Services .....	11
61	4.2	Starting an Agent.....	11
62	4.3	Agent Directory Services.....	11
63	4.3.1	Registering an Agent .....	12
64	4.3.2	Discovering an Agent.....	12
65	4.4	Service Directory Services.....	13
66	4.5	Agent Messages .....	13
67	4.5.1	Message Structure.....	13
68	4.5.2	Message Transport.....	14
69	4.6	Agents Send Messages to Other Agents.....	14
70	4.7	Providing Message Validity and Encryption.....	16
71	4.8	Providing Interoperability.....	17
72	5	Architectural Elements .....	18
73	5.1	Introduction.....	18
74	5.1.1	Classification of Elements.....	18
75	5.1.2	Key-Value Tuples.....	18
76	5.1.3	Services .....	20
77	5.1.4	Format of Element Description .....	20
78	5.1.5	Abstract Elements.....	20
79	5.2	Agent .....	22
80	5.2.1	Summary.....	22
81	5.2.2	Relationships to Other Elements .....	22
82	5.2.3	Description .....	23
83	5.3	Agent Attribute.....	23
84	5.3.1	Summary.....	23
85	5.3.2	Relationships to Other Elements .....	23
86	5.3.3	Description .....	23
87	5.4	Agent Communication Language.....	23
88	5.4.1	Summary.....	23
89	5.4.2	Relationships to Other Elements .....	23
90	5.4.3	Description .....	23
91	5.5	Agent Directory Entry .....	24
92	5.5.1	Summary.....	24
93	5.5.2	Relationships to Other Elements .....	24
94	5.5.3	Description .....	24

95	5.6	Agent Directory Service .....	24
96	5.6.1	Summary .....	24
97	5.6.2	Relationships to Other Elements .....	24
98	5.6.3	Actions .....	24
99	5.6.4	Description .....	26
100	5.7	Agent Locator .....	27
101	5.7.1	Summary .....	27
102	5.7.2	Relationships to Other Elements .....	27
103	5.7.3	Description .....	27
104	5.8	Agent Name .....	27
105	5.8.1	Summary .....	27
106	5.8.2	Relationships to Other Elements .....	27
107	5.8.3	Description .....	2827
108	5.9	Content .....	28
109	5.9.1	Summary .....	28
110	5.9.2	Relationships to Other Elements .....	28
111	5.9.3	Description .....	28
112	5.10	Content Language .....	28
113	5.10.1	Summary .....	28
114	5.10.2	Relationships to Other Elements .....	29
115	5.10.3	Description .....	29
116	5.11	Encoding Representation .....	29
117	5.11.1	Summary .....	29
118	5.11.2	Relationships to Other Elements .....	29
119	5.11.3	Description .....	29
120	5.12	Encoding Service .....	29
121	5.12.1	Summary .....	29
122	5.12.2	Relationships to Other Elements .....	29
123	5.12.3	Actions .....	3029
124	5.12.4	Description .....	31
125	5.13	Envelope .....	31
126	5.13.1	Summary .....	31
127	5.13.2	Relationship to Other Elements .....	31
128	5.13.3	Description .....	31
129	5.14	Explanation .....	31
130	5.14.1	Summary .....	31
131	5.14.2	Relationship to Other Elements .....	31
132	5.14.3	Description .....	32
133	5.15	Message .....	32
134	5.15.1	Summary .....	32
135	5.15.2	Relationships to other elements .....	32
136	5.15.3	Description .....	32
137	5.16	Message Transport Service .....	32
138	5.16.1	Summary .....	32
139	5.16.2	Relationships to Other Elements .....	32
140	5.16.3	Actions .....	32
141	5.16.4	Description .....	34
142	5.17	Ontology .....	34
143	5.17.1	Summary .....	34
144	5.17.2	Relationships to Other Elements .....	34
145	5.17.3	Description .....	34
146	5.18	Payload .....	35
147	5.18.1	Summary .....	35
148	5.18.2	Relationships to Other Elements .....	35
149	5.18.3	Description .....	35
150	5.19	Service .....	35
151	5.19.1	Summary .....	35

152	5.19.2	Relationships to Other Elements .....	35
153	5.19.3	Description .....	35
154	5.20	Service Address .....	35
155	5.20.1	Summary .....	35
156	5.20.2	Relationships to Other Elements .....	36
157	5.20.3	Description .....	36
158	5.21	Service Attributes .....	36
159	5.21.1	Summary .....	36
160	5.21.2	Relationships to Other Elements .....	36
161	5.21.3	Description .....	36
162	5.22	Service Directory Entry .....	36
163	5.22.1	Summary .....	36
164	5.22.2	Relationships to Other Elements .....	36
165	5.22.3	Description .....	37
166	5.23	Services Directory Service .....	37
167	5.23.1	Summary .....	37
168	5.23.2	Relationships to Other Elements .....	37
169	5.23.3	Description .....	37
170	5.23.4	Actions .....	37
171	5.24	Service Id .....	4039
172	5.24.1	Summary .....	4039
173	5.24.2	Relationships to other elements .....	40
174	5.24.3	Description .....	40
175	5.25	Service Location Description .....	40
176	5.25.1	Summary .....	40
177	5.25.2	Relationships to Other Elements .....	40
178	5.25.3	Description .....	40
179	5.26	Service Locator .....	40
180	5.26.1	Summary .....	40
181	5.26.2	Relationships to Other Elements .....	4140
182	5.26.3	Description .....	41
183	5.27	Service Root .....	41
184	5.27.1	Summary .....	41
185	5.27.2	Relationships to Other Elements .....	41
186	5.27.3	Description .....	41
187	5.28	Service Signature .....	41
188	5.28.1	Summary .....	41
189	5.28.2	Relationships to Other Elements .....	41
190	5.28.3	Description .....	41
191	5.29	Service Type .....	42
192	5.29.1	Summary .....	42
193	5.29.2	Relationships to Other Elements .....	42
194	5.29.3	Description .....	42
195	5.30	Signature Type .....	42
196	5.30.1	Summary .....	42
197	5.30.2	Relationships to Other Elements .....	42
198	5.30.3	Description .....	42
199	5.31	Transport .....	4342
200	5.31.1	Summary .....	4342
201	5.31.2	Relationships to Other Elements .....	43
202	5.31.3	Description .....	43
203	5.32	Transport Description .....	43
204	5.32.1	Summary .....	43
205	5.32.2	Relationships to Other Elements .....	43
206	5.32.3	Description .....	43
207	5.33	Transport Message .....	43
208	5.33.1	Summary .....	43

209	5.33.2	Relationships to Other Elements .....	4443
210	5.33.3	Description .....	4443
211	5.34	Transport Specific Address .....	44
212	5.34.1	Summary .....	44
213	5.34.2	Relationships to Other Elements .....	44
214	5.34.3	Description .....	44
215	5.35	Transport Specific Property .....	44
216	5.35.1	Summary .....	44
217	5.35.2	Relationships to Other Elements .....	44
218	5.35.3	Description .....	44
219	5.36	Transport Type .....	4544
220	5.36.1	Summary .....	4544
221	5.36.2	Relationships to Other Elements .....	4544
222	5.36.3	Description .....	45
223	6	Agent and Agent Information Model .....	46
224	6.1	Agent Relationships .....	46
225	6.2	Transport Message Relationships .....	47
226	6.3	Agent Directory Entry Relationships .....	48
227	6.4	Service Directory Entry Relationships .....	49
228	6.5	Message Elements .....	50
229	6.6	Message Transport Elements .....	51
230	7	References .....	52
231	8	Informative Annex A — Goals of Service Model .....	53
232	8.1	Scope .....	53
233	8.2	Variety of Services .....	53
234	8.3	Bootstrapping .....	53
235	8.4	Dynamic services .....	53
236	8.5	Granularity .....	53
237	8.6	Example .....	53
238	9	Informative Annex B — Goals of Message Transport Service Abstraction .....	55
239	9.1	Scope .....	55
240	9.2	Variety of Transports .....	55
241	9.3	Support for Alternative Transports within a Single System .....	55
242	9.4	Desirability of Transport Agnosticism .....	56
243	9.5	Desirability of Selective Specificity .....	56
244	9.6	Connection-Based, Connectionless and Store-and-Forward Transports .....	56
245	9.7	Conversation Policies and Interaction Protocols .....	56
246	9.8	Point-to-Point and Multiparty Interactions .....	56
247	9.9	Durable Messaging .....	57
248	9.10	Quality of Service .....	57
249	9.11	Anonymity .....	57
250	9.12	Message Encoding .....	57
251	9.13	Interoperability and Gateways .....	58
252	9.14	Reasoning about Agent Communications .....	58
253	9.15	Testing, Debugging and Management .....	58
254	10	Informative Annex C — Goals of Directory Service Abstractions .....	59
255	10.1	Scope .....	59
256	10.2	Variety of Directory Services .....	59
257	10.3	Desirability of Directory Agnosticism .....	59
258	10.4	Desirability of Selective Specificity .....	60
259	10.5	Interoperability and Gateways .....	60
260	10.6	Reasoning about Agent Directory .....	60
261	10.7	Testing, Debugging and Management .....	60
262	11	Informative Annex D — Goals for Security and Identity Abstractions .....	61
263	11.1	Introduction .....	61
264	11.2	Overview .....	61
265	11.3	Areas to Apply Security .....	61

266	11.3.1	Content Validity and Privacy During Message Transport.....	61
267	11.3.2	Agent Identity .....	62
268	11.3.3	Agent Principal Validation.....	62
269	11.3.4	Code Signing Validation.....	63
270	11.4	Risks Not Addressed.....	63
271	11.4.1	Code or Data Peeping .....	63
272	11.4.2	Code or Data Alteration .....	63
273	11.4.3	Concerted Attacks.....	63
274	11.4.4	Copy and Replay .....	63
275	11.4.5	Denial of Service.....	63
276	11.4.6	Misinformation Campaigns .....	63
277	11.4.7	Repudiation.....	64
278	11.4.8	Spoofing and Masquerading.....	64
279	11.5	Glossary of Security Terms.....	64
280	12	Informative Annex E — ChangeLog.....	66
281	12.1	2001/11/01 - version I by TC Architecture.....	66
282	12.2	2002/05/02 - version K by FIPA Architecture Board .....	68
283			

## 283 1 Introduction

284 This document, and the specifications that are derived from it, defines the FIPA Abstract Architecture. The parts of the  
285 FIPA abstract architecture include:

- 286
- 287 • A specification that defines architectural elements and their relationships (this document).
- 288
- 289 • Guidelines for the specification of agent systems in terms of particular software and communications technologies  
290 (Guidelines for Instantiation).
- 291
- 292 • Specifications governing the interoperability and conformance of agents and agent systems (Interoperability  
293 Guidelines).
- 294

295 Note that the latter two documents are not yet available.

296 See *Section 2, Scope and Methodology* for a fuller introduction to this document.

298

### 299 1.1 Contents

300 This document is organized into the following sections and a series of annexes.

- 301
- 302 • This **Introduction**.
- 303
- 304 • The **Scope and methodology** section explains the background of this work, its purpose, and the methodology that  
305 has been followed. It describes the role of this work in the overall FIPA work program and discusses both the  
306 current status of the work and way in which the document is expected to evolve.
- 307
- 308 • The **Themes of the Abstract Architecture** section that explains the style and the themes of the Abstract  
309 Architecture specification.
- 310
- 311 • The **Architectural overview** presents an overview of the architecture with some examples. It is intended to  
312 provide the appropriate context for understanding the subsequent sections.
- 313
- 314 • The **Architectural Elements** section comprises the FIPA architecture components.
- 315
- 316 • The **Agent and Agent Information Model** defines UML pattern relationships between **Architectural Elements**.
- 317

318 The annexes include:

- 319
- 320 • **Goals of Service Model**
- 321
- 322 • **Goals of Message Transport Service Abstractions**
- 323
- 324 • **Goals of Directory Service Abstractions**.
- 325
- 326 • **Goals for Security and Identity Abstractions**.
- 327

### 328 1.2 Audience

329 The primary audience for this document is developers of concrete specifications for agent systems – specifications  
330 grounded in particularly technologies, representations, and programming models. It may also be read by the users of  
331 these concrete specifications, including implementers of agent platforms, agent systems, and gateways between agent  
332 systems.

333

334 This document describes an abstract architecture for creating intentional multi-agent systems. It assumes that the  
335 reader has a good understanding about the basic principles of multi-agent systems. It does not provide the background  
336 material to help the reader assess whether multi-agent systems are an appropriate model for their system design, nor  
337 does it provide background material on topics such as Agent Communication Languages, BDI systems, or distributed  
338 computing platforms.

339 The abstract architecture described in this document will guide the creation of concrete specifications of different  
340 elements of the FIPA agent systems. The developers of the concrete specifications must ensure that their work  
341 conform to the abstract architecture in order to provide specifications with appropriate levels of interoperability.  
342 Similarly, those specifying applications that will run on FIPA compliant agent systems will need to understand what  
343 services and features that they can use in the creation of their applications.  
344

### 345 **1.3 Acknowledgements**

346 This document was developed by members of FIPA TC A, the Technical Committee of FIPA charged with this work.  
347 Other FIPA Technical Committees also made substantial contributions to this effort, and we thank them for their effort  
348 and assistance.  
349

350

## 2 Scope and Methodology

This section provides a context for the Abstract Architecture, the scope of the work and methodology employed.

### 2.1 Background

FIPA's goal in creating agent standards is to promote inter-operable agent applications and agent systems. In 1997 and 1998, FIPA issued a series of agent system specifications that had as their goal inter-operable agent systems. This work included specifications for agent infrastructure and agent applications. The infrastructure specifications included an agent communication language, agent services, and supporting management ontologies. There were also a number of application domains specified, such as personal travel assistance and network management and provisioning.

At the heart FIPA's model for agent systems is agent communication, where agents can pass semantically meaningful messages to one another in order to accomplish the tasks required by the application. In 1998 and 1999 it became clear that it would be useful to support variations in those messages:

- How those messages are transferred (that is, the transport).
- How those messages are represented (e.g. s-expressions, bit-efficient binary objects, XML).
- Optional attributes of those messages, such as how to authenticate or encrypt them.

It also became clear that to create agent systems, which could be deployed in commercial settings, it was important to understand and to use existing software environments. These environments included elements such as:

- Distributed computing platforms or programming languages,
- Messaging platforms,
- Security services,
- Directory services, and,
- Intermittent connectivity technologies.

FIPA was faced with two choices: to incrementally revise specifications to add various features such as intermittent connectivity, or to take a more holistic approach. The holistic approach, which FIPA adopted in January of 1999, was to create an architecture that could accommodate a wide range of commonly used mechanisms, such as various message transports, directory services and other commonly, commercially available development platforms. For detailed discussions of the goals of the architecture, see:

- *Section 8, Informative Annex A — Goals of Service Model*
- *Section 9, ~~Informative Annex B — Goals of Message Transport Service Abstraction~~~~Informative Annex B — Goals of Message Transport Service Abstraction~~~~Informative Annex B — Goals of Message Transport Service Abstractions~~*
- *Section 10, Informative Annex C — Goals of Directory Service Abstractions*
- *Section 11, Informative Annex D — Goals for Security and Identity Abstractions*

These describe in greater detail the design considerations that were considered when creating this abstract architecture. In addition, FIPA needed to consider the relationship between the existing FIPA 97, FIPA 98 and FIPA

2000 work and the abstract architecture. While more validation is required, the FIPA 2000 work is in part a concrete realization of this abstract architecture. While one of the goals in creating this architecture was to maintain full compatibility with the FIPA 97 and 98 specifications, this was not entirely feasible, especially when trying to support multiple implementations.

Agent systems built according to FIPA 97 and 98 specifications will be able to inter-operate with agent systems built according to the abstract architecture through transport gateways with some limitations. The FIPA 2000 architecture is a closer match to the abstract architecture, and will be able to fully inter-operate via gateways. The overall goal in this architectural approach is to permit the creation of systems that seamlessly integrate within their specific computing environment while interoperating with agent systems residing in separate environments.

## 2.2 Why an Abstract Architecture?

The first purpose of this work is to foster interoperability and reusability. To achieve this, it is necessary to identify the elements of the architecture that must be codified. Specifically, if two or more systems use different technologies to achieve some functional purpose, it is necessary to identify the common characteristics of the various approaches. This leads to the identification of *architectural abstractions*: abstract designs that can be formally related to every valid implementation.

By describing systems abstractly, one can explore the relationships between fundamental elements of these agent systems. By describing the relationships between these elements, it becomes clearer how agent systems can be created so that they are interoperable. From this set of architectural elements and relations one can derive a broad set of possible concrete architectures, which will interoperate because they share a common abstract design.

Because the abstract architecture permits the creation of multiple concrete realizations, it must provide mechanisms to permit them to interoperate. This includes providing transformations for both transport and encodings, as well as integrating these elements with the basic elements of the environment.

For example, one agent system may transmit ACL messages using the OMG IIOP protocol. A second may use IBM's MQ-series enterprise messaging system. An analysis of these two systems – how senders and receivers are identified, and how messages are encoded and transferred – allows us to arrive at a series of architectural abstractions involving messages, encodings, and addresses.

## 2.3 Scope of the Abstract Architecture

The primary focus of this abstract architecture is to create semantically meaningful message exchange between agents which may be using different messaging transports, different Agent Communication Languages, or different content languages. This requires numerous points of potential interoperability. The scope of this architecture includes:

- A model of services and discovery of services available to agents and other services.
- Message transport interoperability.
- Supporting various forms of ACL representations.
- Supporting various forms of content language.
- Supporting multiple directory services representations.

It must be possible to create implementations that vary in some of these attributes, but which can still interoperate. Some aspects of potential standardization are outside of the scope of this architecture. There are three different reasons why things are out of scope:

- The area cannot be described abstractly.

- 455 • The area is not yet ready for standardization, or there was not yet sufficient agreement about how to standardize it.  
 456  
 457 • The area is sufficiently specialized that it does not currently need standardization.  
 458

459 Some of the key areas that are **not** included in this architecture are:

- 460  
 461 • Agent lifecycle and management.  
 462  
 463 • Agent mobility.  
 464  
 465 • Domains.  
 466  
 467 • Conversational policy.  
 468  
 469 • Agent Identity.  
 470

471 The next sections describe the rationale for this in more detail. However, it is extremely important to understand that  
 472 the abstract architecture does not prohibit additional features – it merely addresses how interoperable features should  
 473 be implemented. It is anticipated that over time some of these areas will be part of the interoperability of agent  
 474 systems.  
 475

### 476 **2.3.1 Areas that are not Sufficiently Abstract**

477 An abstraction may not appear in the abstract architecture because there is no clean abstraction for different models  
 478 of implementation. Two examples of this are agent lifecycle management and security related issues.  
 479

480 For example, in examining agent lifecycle, it seems clear there are a minimum set of features that are required:  
 481 Starting an agent, stopping an agent, "freezing" or "suspending" an agent, and "unfreezing" or "restarting" an agent. In  
 482 practice, when one examines how various software systems work, very little consistency is detected inside the  
 483 mechanisms, or in how to address and use those mechanisms. Although it is clear that concrete specifications will  
 484 have to address these issues, it is not clear how to provide a unifying abstraction for these features. Therefore there  
 485 are some architectural elements that can only appear at the concrete level, because the details of different  
 486 environments are so diverse.  
 487

488 Security has similar issues, especially when trying to provide security in the transport layer, or when trying to provide  
 489 security for attacks that can occur because a particular software environment has characteristics that permits that sort  
 490 of attack. Agent mobility is another implementation specific model that cannot easily be modelled abstractly.  
 491

492 Both of these topics will be addressed in the *Instantiation Guidelines*, because they are an important part of how agent  
 493 systems are created. However, they cannot be modelled abstractly, and are therefore not included at the *abstract* level  
 494 of the architecture.  
 495

### 496 **2.3.2 Areas for Future Consideration**

497 FIPA may address a number of areas of agent standardization in the future. These include ontologies, domains,  
 498 conversational policies and mechanisms that are used to control systems (resource allocation and access control  
 499 lists), and agent identity. These all represent ideas requiring further development.  
 500

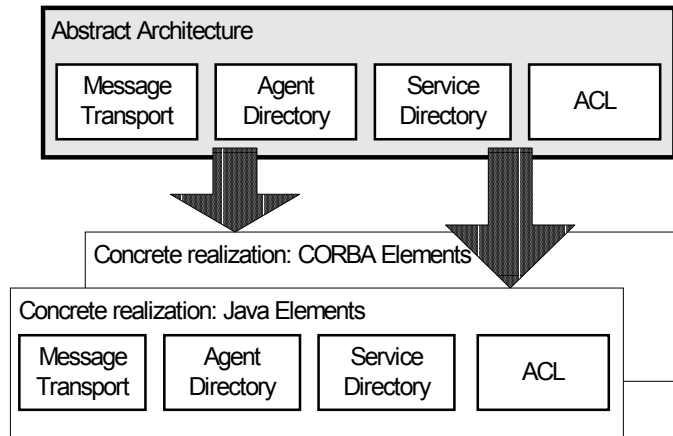
501 This architecture does not address application interoperability. The current model for application interoperability is that  
 502 agents that communicate using a shared set of semantics (such as represented by an ontology) can potentially  
 503 interoperate. This architecture does not extend this model any further.  
 504

## 2.4 Going From Abstract to Concrete Specifications

This document describes an abstract architecture. Such an architecture cannot be directly implemented, but instead the forms the basis for the development of concrete architectural specifications. Such specifications describe in precise detail how to construct an agent system, including the agents and the services that they rely upon, in terms of concrete software artefacts, such as programming languages, applications programming interfaces, network protocols, operating system services, and so forth.

In order for a concrete architectural specification to be FIPA compliant, it must have certain properties. First, the concrete architecture must include mechanisms for agent registration and agent discovery and inter-agent message transfer. These services must be explicitly described in terms of the corresponding elements of the FIPA abstract architecture. The definition of an abstract architectural element in terms of the concrete architecture is termed a *realization* of that element; more generally, a concrete architecture will be said to *realize* all or part of an abstraction.

The designer of the concrete architecture has considerable latitude in how he or she chooses to realize the abstract elements. If the concrete architecture provides only one encoding for messages, or only one transport protocol, the realization may simplify the programmatic view of the system. Conversely, a realization may include additional options or features that require the developer to handle both abstract and platform-specific elements. That is to say that the existence of an abstract architecture does not *prohibit* the introduction of elements useful to make a good agent system, it merely sets out the *minimum* required elements.



**Figure 1:** Abstract Architecture Mapped to Various Concrete Realizations

525  
526  
527  
528

The abstract architecture also describes *optional* elements. Although an element is optional at the abstract level, it may be *mandatory* in a particular realization. That is, a realization may require the existence of an entity that is optional at the abstract level (such as a **message-transport-service**), and further specify the features and interfaces that the element must have in that realization.

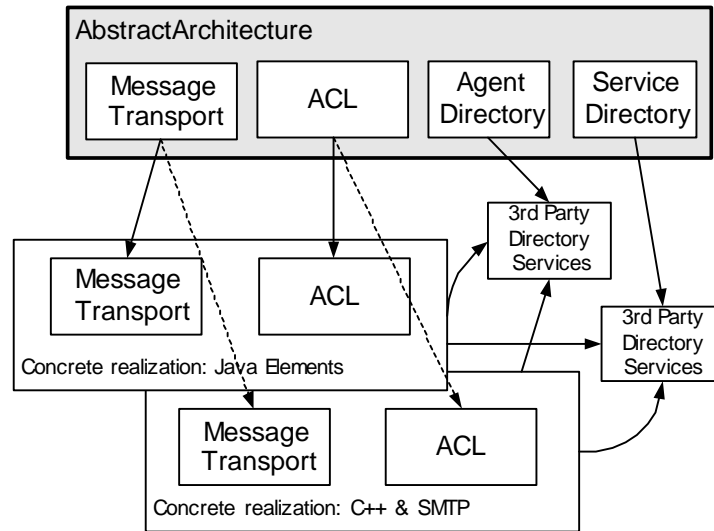
533

It is also important to note that a realization can be of the entire architecture, or just one element. For example, a series of concrete specifications could be created that describe how to represent the architecture in terms of particular programming language, coupled to a sockets-based message transport. Messages are handled as objects with that language, and so on.

538

On the other hand, there may be a single element that can be defined concretely, and then used in a number of different systems. For example, if a concrete specification were created for the **agent-directory-service** element that describes the schemas to use when implemented in LDAP, that particular element might appear in a number of different agent systems.

543



**Figure 2:** Concrete Realizations Using a Shared Element Realization

In this example, the concrete realization of directory is to implement the directory services in LDAP. Several realizations have chosen to use this directory service model.

## 2.5 Methodology

This abstract architecture was created by the use of UML modelling, combined with the notions of design patterns as described in [Gamma95]. Analysis was performed to consider a variety ways of structuring software and communications components in order to implement the features of an intelligent multi-agent system. This ideal agent system was to be capable of exhibiting execution autonomy and semantic interoperability based on an intentional stance. The analysis drew upon many sources:

- The abstract notions of agency and the design features that flow from this.
- Commercial software engineering principles, especially object-oriented techniques, design methodologies, development tools and distributed computing models.
- Requirements drawn from a variety of applications domains.
- Existing FIPA specifications and implementations.
- Agent systems and services, including FIPA and non-FIPA designs.
- Commercially important software systems and services, such as Java, CORBA, DCOM, LDAP, X.500 and MQ Series.

The primary purpose of this work is to foster interoperability and reusability. To achieve this, it is necessary to identify the elements of the architecture that must be codified. Specifically, if two or more systems use different technologies to achieve some functional purpose, it is necessary to identify the common characteristics of the various approaches. This leads to the identification of *architectural elements*: abstract designs that can be formally related to every valid implementation.

For example, one agent system may transmit ACL messages using the OMG IOP protocol. A second may use IBM's MQ-series enterprise messaging system. An analysis of these two systems – how senders and receivers are identified, and how messages are encoded and transferred – allows us to arrive at a series of architectural abstractions involving messages, encodings, and addresses.

581

582 In some areas, the identification of common abstractions is essential for successful interoperation. This is particularly  
583 true for agent-to-agent message transfer. The end-to-end support of a common agent communication language is at  
584 the core of FIPA's work. These essential elements, which correspond to mandatory implementation specifications are  
585 here described as *mandatory architectural elements*. Other areas are less straightforward. Different software systems,  
586 particularly different types of commercial middleware systems, have specialized frameworks for software deployment,  
587 configuration, and management, and it is hard to find common principles. For example, security and identity remain  
588 tend to be highly dependent on implementation platforms. Such areas will eventually be the subjects of architectural  
589 specification, but not all systems will support them. These architectural elements are *optional*.

590

591 This document models the elements and their relationships. In *Section 3, Themes of the Abstract Architecture* there is  
592 an holistic overview of the architecture. In *Section 4, Architectural Overview* there is a structural overview of the  
593 architecture. In *Section 5, Architectural Elements*, each of the architectural elements is described. In *Section 6, Agent  
594 and Agent Information Model* there are diagrams in UML notation to describe the relationships between the elements.  
595

## 596 **2.6 Status of the Abstract Architecture**

597 There are several steps in creating the abstract architecture:

598

599 1. Modelling of the abstract elements and their relationships.

600

601 2. Representing the other requirements on the architecture that cannot be modelled abstractly.

602

603 3. Describing interoperability points.

604

605 This document represents the first item in the list. It is nearing completion, and ready for review.

606

607 The second step is satisfied by *guidelines for instantiation*. This document will not be written until at least one  
608 implementation based on the abstract architecture has been created, as it is desirable to base such a document on  
609 actual implementation experience.

610

611 Interoperability points and conformance are defined by specific *interoperability profiles*. These profiles will be created  
612 as required during the creation of concrete specifications.  
613

## 614 **2.7 Evolution of the Abstract Architecture**

615 One of the challenges involved in creating this specification was drawing the line between elements that belong in the  
616 abstract architecture and those which belong in concrete instantiations of the architecture. As FIPA creates several  
617 concrete specifications, and explores the mechanisms required to properly manage interoperation of these  
618 implementations, some features of the concrete architectures may be abstracted and incorporated in the FIPA abstract  
619 architecture. Likewise, certain abstract architectural elements may eventually be dropped from the abstract  
620 architecture, but may continue to exist in the form of concrete realizations.

621

622 The current placement of various elements as mandatory or optional is somewhat tentative. It is possible that some  
623 elements that are currently optional will, upon further experience in the development of the architecture become  
624 mandatory.

625

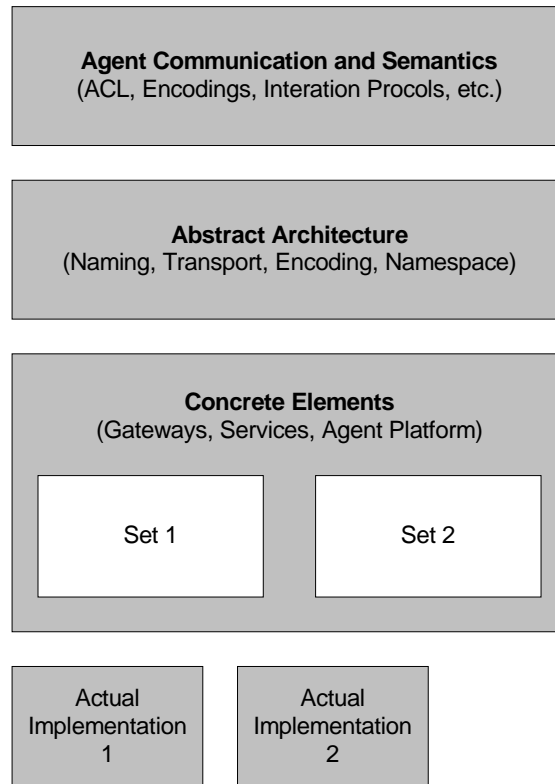
626

### 3 Themes of the Abstract Architecture

The overall approach of the abstract architecture is deeply rooted in object-oriented design, including the use of design patterns and UML modelling. As such, the natural way to envision the elements of the architecture is as a set of abstract object classes that can act as the input to the high level design of specific implementations.

Although the architecture explicitly avoids any specific model of composing its elements, its natural expression is a set of object classes comprising an agent platform that supports agents and services.

The following diagram depicts the hierarchical relationships between the abstraction defined by this document and the elements of a specific instantiation:



**Figure 3:** Relationship between Abstract and Concrete Architecture Elements

Several themes pervade the architecture; these capture the interaction between elements and their intended use.

The first theme is of opaque typed elements, which can be understood by specific implementations of a service. For example, the details of each transport description are opaque to other layers of the system. The transport descriptor provides a transport type, such as *fipa-tcpip-raw-socket* which acts to select the specific transport service that can interpret the transport-specific-address. Thus, a given address element, opaque to other portions of the system, might be *foo.bar.baz.com:1234* which would be readily understood by the above transport service. Opaque typed elements are used in both message encoding and directory services.

This theme leads to an elegant solution for extensibility. Additional implementations of a service may be dynamically added to an environment by defining a new opaque typed element and associating it with the new service. For example, it may be required that a transport mechanism such as the Simple Object Access Protocol (SOAP) be supported within the environment. The transport type ontology would be extended to include a new term, *fipa-soap-v1*. Note that this resembles a polymorphic type scheme.

656 A second repeated theme is the creation of an association (in the form of a contract) between an agent and a service,  
657 such that the agent may then use the service through a returned handle. Note that this theme is intentionally well  
658 suited for implementation through the factory design patterns.

659  
660 For those familiar with the "design pattern" approach to describing system structure, these themes may be naturally  
661 implemented using the factory pattern.  
662

### 663 **3.1 Focus on Agent Interoperability**

664 The Abstract Architecture focuses on core interoperability between agents. These include:

- 665 • Managing multiple message transport schemes,
- 666 • Managing message encoding schemes, and,
- 667 • Locating agents and services via directory services.

668  
669  
670  
671 The Abstract Architecture explicitly avoids issues internal to the structure of an agent. It also largely defers details of  
672 agent services to more concrete architecture documents.  
673

674  
675 After reading through the abstract architecture, many readers may feel that it lacks a number of elements they would  
676 have expected to be included. Examples include the notion of an "agent-platform," "gateways" between agent systems,  
677 bootstrapping of agent systems and agent configuration and coordination.  
678

679 These elements are not included in the abstract architecture because they are inherently coupled with specific  
680 implementations of the architecture, rather than across all possible implementations. The forthcoming document  
681 "Concrete Architectural Elements" will describe many of these elements in terms of specific environments. Beyond this,  
682 some elements will exist only in specific instantiations.  
683

### 684 **3.2 An Exemplar System**

685 In order to further illuminate the intended use of the architectural elements, let us consider an agent platform,  
686 implemented in an object oriented environment. The system uses the components of the abstract architecture to  
687 implement two separate object factories; a transport factory and an encoding factory. A directory service is also  
688 provided, with access through a static object.  
689

690 Agents in the environment are constructed as objects, each running on a permanent thread. Each has access to the  
691 two agent factories, as well as the directory service.  
692

693 When an agent wants to send a message to another agent, it uses the directory service to obtain a set of transport-  
694 descriptors for the agent. It then passes these transport-descriptors to the transport factory, which returns a transport-  
695 handle. It should be noted that the transport factory and handle are not parts of the abstract architecture, but rather  
696 artefacts of the specific implementation. The agent then uses an encoder provided by the encoding factory, to  
697 transform the message into the desired encoding. Finally it transfers this encoded message to the recipient via the  
698 selected transport.  
699

## 699 4 Architectural Overview

700 The FIPA architecture defines at an abstract level how two agents can locate and communicate with each other by  
 701 registering themselves and exchanging messages. To do this, a set of architectural elements and their relationships  
 702 are described. In this section the basic relationships between the elements of the FIPA agent system are described. In  
 703 *Section 5, Architectural Elements* and *Section 6, Agent and Agent Information Model*, there are descriptions of each  
 704 element (including mandatory or optional status) and UML Models for the architecture, respectively.

705  
 706 This section gives a relatively high level description of the notions of the architecture. It does not explain all of the  
 707 aspects of the architecture. Use this material as an introduction, which can be combined with later sections to reach a  
 708 fuller understanding of the abstract architecture.

709

### 710 4.1 Agents and Services

711 **Agents** communicate by exchanging messages which represent speech acts, and which are encoded in an **agent-**  
 712 **communication-language**.

713

714 **Services** provide support services for **agents**. In addition to a number of standard services including **agent-directory-**  
 715 **services** and **message-transport-services** this version of the Abstract Architecture defines a general service model  
 716 that includes a **service-directory-service**.

717

718 The Abstract architecture is explicitly neutral about how **services** are presented. They may be implemented either as  
 719 **agents** or as software that is accessed via method invocation, using programming interfaces such as those provided in  
 720 Java, C++, or IDL. An **agent** providing a **service** is more constrained in its behaviour than a general-purpose agent. In  
 721 particular, these agents are required to preserve the semantics of the service. This implies that these agents do not  
 722 have the degree of autonomy normally attributed to agents. They may not arbitrarily refuse to provide the service.

723

724 It should be noted that if **services** are implemented as **agents** there are potential problems that may arise with  
 725 discovering and communicating with these services. The resolution of these issues is beyond the scope of this  
 726 document.

727

### 728 4.2 Starting an Agent

729 On start-up an agent must be provided with a **service-root**. Typically the provider of the **service-root** will be a  
 730 **service-directory-service** which will supply a set of **service-locators** for available agent lifecycle support services,  
 731 such as **message-transport-services**, **agent-directory-services** and **service-directory-services**. In general, a  
 732 **service-root** will provide sufficient entries to either describe all of the services available within the environment  
 733 directly, or it will provide pointers to further services which will describe these services.

734

### 735 4.3 Agent Directory Services

736 The basic role of the **agent-directory-service** is to provide a location where **agents** register their descriptions as  
 737 **agent-directory-entries**. Other **agents** can search the **agent-directory-entries** to find **agents** with which they wish to  
 738 interact.

739

740 The **agent-directory-entry** is a **key-value-tuple** consisting of at least the following two **key-value-pairs**:

741

<b>Agent-name</b>	A globally unique name for the <b>agent</b>
<b>Agent-locator</b>	One or more <b>transport-descriptions</b> , each of which is a self describing structure containing a <b>transport-type</b> , a <b>transport-specific-address</b> and zero or more <b>transport-specific-properties</b> used to communicate with the <b>agent</b>

742

In addition the **agent-directory-entry** may contain other descriptive attributes, such as the services offered by the **agent**, cost associated with using the **agent**, restrictions on using the **agent**, etc.

Note that the keys **agent-name** and **agent-locator** are short-form for the fully qualified names in the FIPA controlled namespace. See *Section 5.1.2, Key-Value Tuples* for further details.

### 4.3.1 Registering an Agent

Agent A wishes to advertise itself as a provider of some service. It first binds itself to one or more **transports**. In some implementations it will delegate this task to the **message-transport-service**; in others it will handle the details of, for example, contacting an ORB, or registering with an RMI registry, or establishing itself as a listener on a message queue. As a result of these actions, the agent is addressable via one or more **transports**.

Having established bindings to one or more **message-transport-services** the agent must advertise its presence. The agent realizes this by constructing an **agent-directory-entry** and registering it with the **agent-directory-service**. The **agent-directory-entry** includes the **agent-name**, its **agent-locator** and optional attributes that describe the service. For example, a stock service might advertise itself in abstract terms as {agent-service, "com.dowjones.stockticker"} and {ontology, org.fipa.ontology.stockquote}<sup>1</sup>.

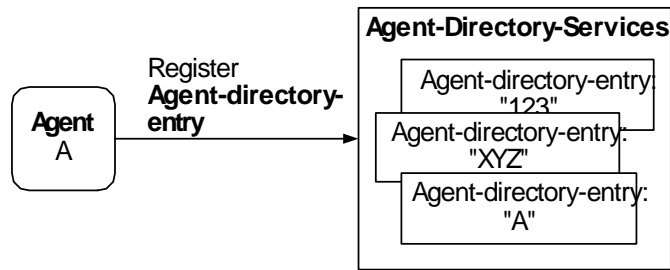
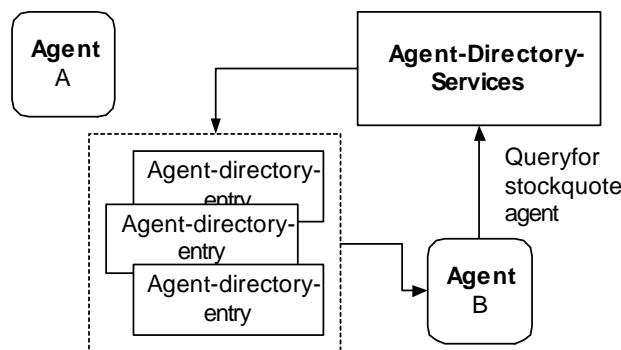


Figure 4: An Agent Registers with a Directory Service

### 4.3.2 Discovering an Agent

Agents can use the **agent-directory-service** to locate other agents with which to communicate. With reference to Figure 5, if agent B is seeking stock quotes, it may search for an agent that advertises use of the stockquote ontology. Technically, this would involve searching for an **agent-directory-entry** that includes the **key-value-pair** {ontology, {com, dowjones, ontology, stockquote}}. If it succeeds it will retrieve the **agent-directory-entry** for agent A. It might also retrieve other **agent-directory-entries** for agents that support that ontology.



<sup>1</sup> Note that the quoted string in the first example is a quoted value whereas the other elements are abstract names represented as tuples that may be encoded in a variety of different ways.

Figure 5: Directory Query

Agent B can then examine the returned **agent-directory-entries** to determine which agent best suits its needs. The **agent-directory-entries** include the **agent-name**, the **agent-locator**, which contains information related to how to communicate with the agent, and other optional attributes.

#### 4.4 Service Directory Services

The basic role of the **service-directory-service** is to provide a consistent means by which agents and services can discover services. Operationally, the **service-directory-service** provides a location where **services** can register their service descriptions as **service-directory-entries**. Also, **agents** and **services** can search the **service-directory-service** to locate services appropriate to their needs.

The **service-directory-service** is analogous to but different to the **agent-directory-services**; the latter are oriented towards discovering **agents** whereas the former is oriented to discovering **services**. In practice also, the two kinds of directories may have radically different realizations. For example, on some systems a **service-directory-service** may be modelled simply as a fixed table of a small size whereas the **agent-directory-service** may be modelled using LDAP or other distributed directory technologies.

The entries in a **service-directory-service** are service descriptions consisting of a tuple containing a **service-id**, **service-name**, **service-type**, a **service-locator** and a set of optional **service-attributes**. The **service-locator** is a typed structure that may be used by **services** and **agents** to access the service.

The **service-directory-entry** is a **key-value-tuple** consisting of at least the following **key-value-pairs**:

<b>Service-nameid</b>	A globally unique name for the <b>service</b>
<b>Service-type</b>	The categorized <i>type</i> of the <b>service</b>
<b>Service-locator</b>	One or more <b>key-value tuples</b> containing a <b>signature type</b> , <b>service signature</b> and <b>service address</b> each

Additional **service-attributes** may be included that contain other descriptive properties of the **service**, such as the cost associated with using the **service**, restrictions on using the **service**, etc.

As a foundation for bootstrapping, each realization of the **service-directory-service** will provide agents with a **service-root**, which will take the form of a set of **service-locators** including at least one **service-directory-service**. (pointing to itself).

#### 4.5 Agent Messages

In FIPA agent systems agents communicate with one another, by sending messages. Three fundamental aspects of message communication between agents are the message structure, message representation and message transport.

##### 4.5.1 Message Structure

The structure of a **message** is a **key-value-tuple** (see *Section 5.1.2, Key-Value Tuples*) and is written in an **agent-communication-language**, such as FIPA ACL. The **content** of the **message** is expressed in a **content-language**, such as KIF or SL. **Content** expressions can be grounded by ontologies referenced within the **ontology key-value-tuple**. The messages also contain the **sender** and **receiver** names, expressed as **agent-names**. **Agent-names** are unique name identifiers for an agent. Every message has one sender and zero or more receivers. The case of zero receivers enables broadcasting of messages such as in ad-hoc wireless networks.

**Messages** can recursively contain other messages.

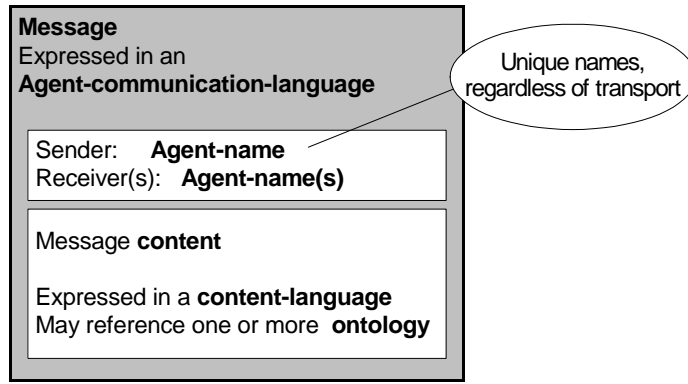


Figure 6: A Message

822  
823  
824  
825

826 **4.5.2 Message Transport**

827 When a **message** is sent it is encoded into a **payload**, and included in a **transport-message**. The **payload** is  
828 encoded using the **encoding-representation** appropriate for the transport. For example, if the **message** is going to be  
829 sent over a low bandwidth transport (such a wireless connection) a bit efficient representation may used instead of a  
830 string representation to allow more efficient transmission.

831  
832 The **transport-message** itself is the **payload** plus the **envelope**. The **envelope** includes the sender and receiver  
833 **transport-descriptions**. The **transport-descriptions** contain the information about how to send the message (via  
834 what transport, to what address, with details about how to utilize the transport). The **envelope** can also contain  
835 additional information, such as the **encoding-representation**, data related security, and other realization specific data  
836 that needs be visible to the **transport** or recipient(s).

837

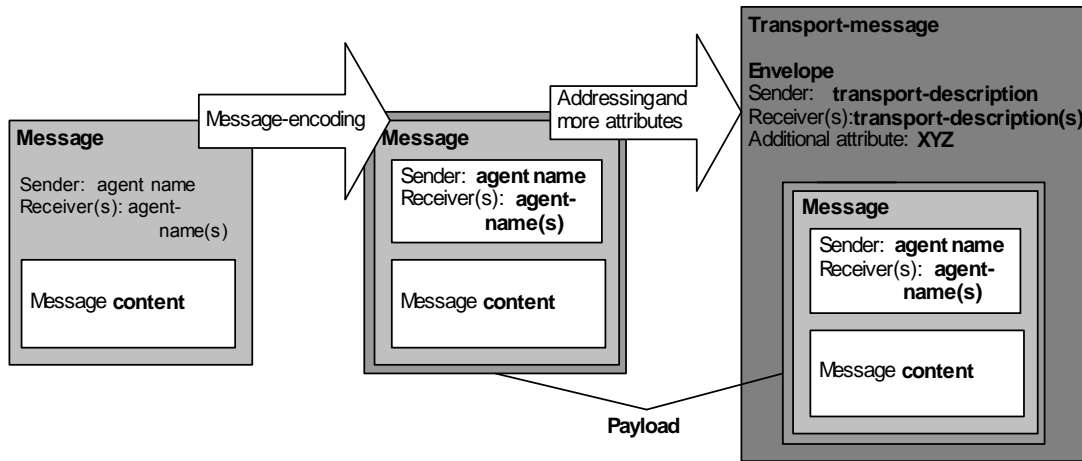


Figure 7: A Message Becomes a Transport-message

838  
839  
840  
841  
842  
843  
844  
845  
846  
847

In the above diagram, a **message** is encoded into a **payload** suitable for transport over the selected **message-transport**. It should be noted that **payload** adds nothing to the message, but only encodes it into another representation. An appropriate **envelope** is created that has sender and receiver information that uses the **transport-description** data appropriate to the transport selected. There may be additional envelope data also included. The combination of the payload and envelope is termed as a **transport-message**.

848 **4.6 Agents Send Messages to Other Agents**

849 In FIPA agent systems agents are intended to communicate with one another. Hence, here are some of the basic  
850 notions about agents and their communications:

851  
852  
853  
854  
855  
856  
857  
858  
859  
860  
861  
862  
863  
864  
865  
866  
867  
868  
869  
870  
871  
872  
873  
874  
875  
876  
877  
878  
879  
880  
881

Each **agent** has an **agent-name**. This **agent-name** is unique and unchangeable. Each agent also has one or more **transport-descriptions**, which are used by other agents to send a **transport-message**. Each **transport-description** correlates to a particular form of message **transport**, such as IOP, SMTP, or HTTP. A **transport** is a mechanism for transferring messages. A **transport-message** is a message that sent from one agent to another in a format (or encoding) that is appropriate to the **transport** being used. A set of **transport-descriptions** can be held in an **agent-locator**.

For example, there may be an **agent** with the **agent-name** "ABC". This agent is addressable through two different transports, HTTP and SMTP. Therefore, the agent has two **transport-descriptions**, which are held in the **agent-locator**. The transport descriptions are as follows:

**Directory entry for ABC**

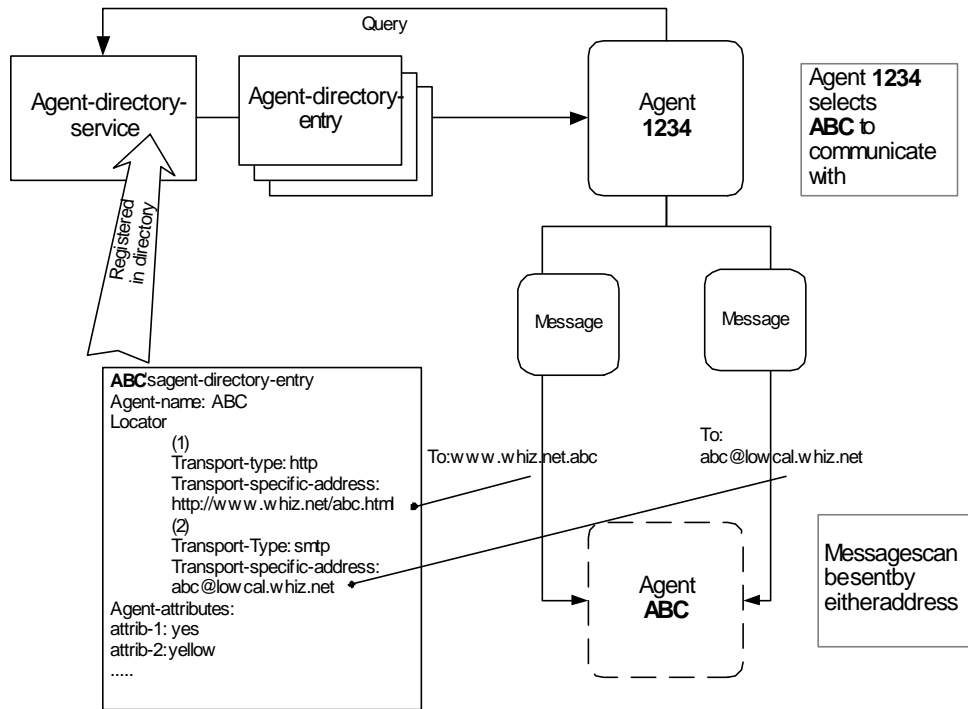
Agent-name: ABC  
Agent Locator:

Transport-type	Transport-specific-address	Transport-specific-property
HTTP	http://www.whiz.net/abc	(none)
SMTP	Abc@lowcal.whiz.net	(none)

Agent-attributes:      Attrib-1: yes  
                          Attrib-2: yellow  
                          Language: French, German, English  
                          Preferred negotiation: contract-net

Note: in this example, the **agent-name** is used as part of the **transport-descriptions**. This is just to make these examples easier to read. There is *no* requirement to do this.

Another agent can communicate with agent "ABC" using either **transport-description**, and thereby know which agent it is communicating with. In fact, the second agent can even change transports and can continue its communication. Because the second agent knows the **agent-name**, it can retain any reasoning it may be doing about the other agent, without loss of continuity.

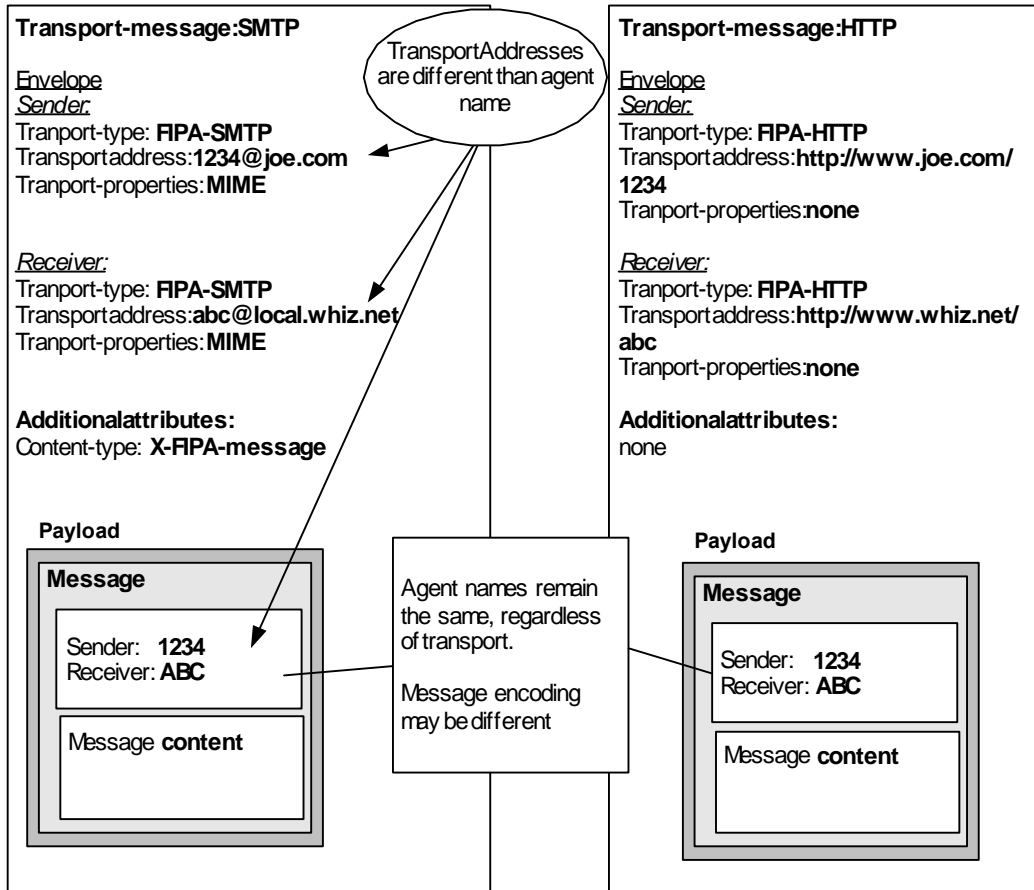


**Figure 8: Communicating Using Any Transport**

882  
883  
884  
885  
886  
887  
888  
889

In the above diagram, Agent 1234 can communicate with Agent ABC using either an SMTP transport or an HTTP transport. In either case, if Agent 1234 is doing any reasoning about agents that it communicates with, it can use the **agent-name** "ABC" to record which agent it is communicating with, rather than the transport description. Thus, if it changes transports, it would still have continuity of reasoning.

Here's what the messages on the two different transports might look like:



890  
891  
892  
893  
894  
895  
896  
897

Figure 9: Two Transport-Messages to the Same Agent

In the diagram above, the **transport-description** is different, depending on the transport that is going to be used. Similarly, the **message-encoding** of the **payload** may also be different. However, the **agent-names** remain consistent across the two message representations.

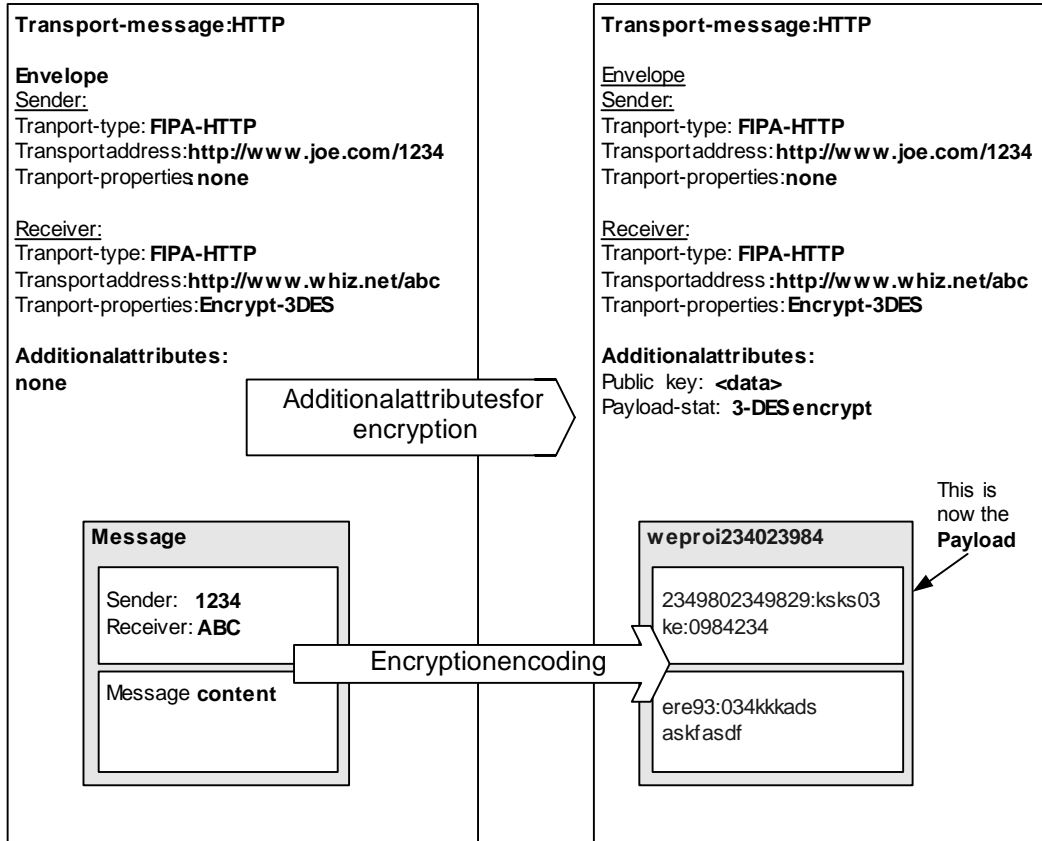
#### 4.7 Providing Message Validity and Encryption

There are many aspects of security that can be provided in agent systems. See *Section 11, Informative Annex D — Goals for Security and Identity Abstractions* for a discussion of possible security features. In this abstract architecture, there is a simple form of security: message validity and message encryption. In message validity, messages can be sent in such a way that any modification during transmission is identifiable. In message encryption, a message is sent in encrypted form such that non-authorized entities cannot comprehend the message content.

In the abstract architecture these features are accommodated through **encoding-representations** and the use of additional attributes in the **envelope**. For example, as the payload is encoded, one of the encodings could be to a digitally encrypted set of data, using a public key and preferred encryption algorithm. Additional parameters are added to the envelope to indicate these characteristics.

908

909



910  
911  
912  
913  
914  
915  
916

**Figure 10:** Encrypting a Message Payload

In the above diagram, the payload is encrypted, and additional attributes added to the envelope to support the encryption. These attributes must remain unencrypted in order that the receiving party is able to use them.

917 **4.8 Providing Interoperability**

918 There are two ways in which the abstract architecture makes provision for interoperability. The first is **transport**  
919 interoperability. The second is **message** representation interoperability.

920  
921 To provide interoperability, there are certain elements that must be included throughout the architecture to permit  
922 multiple implementations. For example, earlier it was noted that an **agent** has both an **agent-name** and an **agent-**  
923 **locator**. The **locator** contains **transport-descriptions**, each of which contains information necessary for a particular  
924 transport to send a message to the corresponding agent. The semantics of agent communication require that an  
925 agent's name be preserved throughout its lifetime, regardless of what transports may be used to communicate with it.

926  
927

## 927 5 Architectural Elements

928 The elements of the abstract architecture are defined here. For each element, the semantics are described informally  
 929 followed by the relationships between the element and others.

### 931 5.1 Introduction

#### 932 5.1.1 Classification of Elements

933 The word **element** is used here to indicate an item or entity that is part of the architecture, and participates in  
 934 relationships with other elements of the architecture.

935  
 936 The architectural elements are classified as *mandatory* or *optional*. Mandatory elements must appear in all  
 937 instantiations of the FIPA abstract architecture. They describe the fundamental services, such as agent registration  
 938 and communications. These elements are the core aspects of the architecture. Optional elements are not mandatory;  
 939 they represent architecturally useful features that may be shared by some, but not all, concrete instantiations. The  
 940 abstract architecture only defines those optional elements that are highly likely to occur in multiple instantiations of the  
 941 architecture.

942  
 943 These descriptors and classifications are summarised in *Table 1*.

944

Word	Definition
<b>Can, May</b>	In relationship descriptions, the word can or may is used to indicate this is an optional relationship. For example, a <b>service</b> <i>may</i> provide an API invocation, but it is not required to do so.
<b>Element, or architectural element</b>	A member of this abstract architecture. The word <b>element</b> is used here to indicate an item or entity that is part of the architecture, and participates in relationships with other elements of the architecture.
<b>Mandatory</b>	Description of an element or relationship. Required in all fully functional implementations of the FIPA Abstract Architecture.
<b>Must</b>	In relationship descriptions, the word must is used to indicate this is a mandatory relationship. For example, an <b>agent</b> <i>must</i> have an <b>agent-name</b> means that an <b>agent</b> is required to have an <b>agent-name</b> .
<b>Optional</b>	Description of an element or relationship. May appear in any implementation of the FIPA Abstract Architecture, but is not required. Functionality that is common enough that it was included in model.
<b>Realize, realization</b>	To create a concrete specification or instantiation from the abstract architecture. For example, there may be a design to implement the abstract notion of <b>agent-directory-services</b> in LDAP. This could also be said that there is a <i>realization</i> of <b>agent-directory-services</b> .
<b>Relationship</b>	A connection between two elements in the architecture. The relationship between two elements is named (for example "is an instance of", "sends message to") and may have other attributes, such as whether it is required, optional, one-to-one, or one-to-many. The term as used in this document, is very much the way the term is used in UML or other system modelling techniques.

945

946

947

**Table 1:** Terminology

#### 948 5.1.2 Key-Value Tuples

949 Many of the elements of the abstract architecture are defined to be **key-value-tuples**, or **KVTs**. For example, an ACL  
 950 message, its envelope, and agent descriptions are all KVTs. The concept of a **KVT** is central to the notion of  
 951 architectural extensibility, and so it is discussed in some length here.

952

953 A **KVT** consists of an unordered set of **key-value-pairs**. Each **key-value-pair** has two elements, as the term implies.  
 954 The first element, the **key**, is a **pair-element** drawn from an administered name space. All keys defined by the Abstract  
 955 Architecture are drawn from a name space managed by FIPA. This makes it possible for concrete architectures, or  
 956 individual implementations, to add new architectural elements in a manner which is guaranteed not to conflict with the  
 957 Abstract Architecture. The second element of the **key-value-pair** is the **value**. The type of value depends on the **key**.  
 958 In many cases, the value is another **pair-element**, an identifier drawn from a name-space. In other cases, the **value** is  
 959 a constant or expression of some specific type.

960  
 961 The rest of this section describes the rules governing the names for **keys** and **values**.  
 962

963 Traditionally, **pair-elements** have been treated as simple text strings. It is more useful to adopt a more abstract model  
 964 in which abstract identifiers and keywords may be encoded in a variety of different ways.  
 965

966 It is also important that the FIPA elements represented as **key-value-tuples** should be extensible. There are three  
 967 types of extension that can be envisaged:  
 968

- 969 • Official FIPA sanctioned standard extensions,
- 970
- 971 • Durable vendor-specific extensions, and,
- 972
- 973 • Temporary, probably private, extensions.

974  
 975 The last of these has traditionally been addressed by using a particular prefix string ("X-").  
 976

977 Every **pair-element** is an ordered tuple of **tokens**. This tuple denotes a name within a hierarchical namespace, in  
 978 which the first **token** in the tuple is at the highest level in the hierarchy and the rightmost is the leaf. Examples of tuples  
 979 are:  
 980

```
981     {org, fipa, standard, ontology, foo}
982     {com, sun, java, agent, performative, brainwash}
983     {x, cc}
984     {protocol}
```

985  
 986 A **pair-element** containing more than one **token** is a **qualified-element**. In a **qualified-element**, the left-most **token**  
 987 must correspond to one of the top-level ICANN domain names, or to an **anonymous-token**. The latter is used to  
 988 introduce temporary, experimental **qualified-elements**.  
 989

990 With reference to the FQN (Fully Qualified Name) field in Table 2, if a **pair-element** contains only one **token**, it is an  
 991 **unqualified-element**. An **unqualified-element** is interpreted according to Table 2, as though its **token** were  
 992 appended to a tuple of tokens defining a FIPA standard name space, as follows:  
 993

994 For example, the **pair-element**

```
995     { {ontology}, {foo} }
```

996  
 997 is equivalent to,

```
998     { {org, fipa, standard, message, ontology}, {org, fipa, standard, message, ontology, foo} }
```

1000  
 1001 The natural encoding of a **pair-element** is as a sequence of text strings separated by dots. Thus the **pair-element**

```
1002     { {org, fipa, standard, message, ontology}, {org, fipa, standard, message, ontology, foo} },
```

1003  
 1004 will naturally be encoded as:  
 1005

```
1006     org.fipa.standard.message.ontology org.fipa.standard.message.ontology.foo
```

1009

1010 **5.1.3 Services**

1011 A **service** is defined in terms of a set of **actions** that it supports. Each action defines an interaction between the  
 1012 **service** and the **agent** using the service. The semantics of these actions are described informally, to minimize  
 1013 assumptions about how they might be reified in a concrete specification.  
 1014

1015 **5.1.4 Format of Element Description**

1016 The architectural elements are described below. The format of the description is:

1017

1018 • **Summary.** A summary of the element.

1019

1020 • **Relationship to other elements.** A complete description of the relationship of this element to the other  
 1021 architectural elements.

1022 • **Actions.** In the case of mandatory services, the actions that may be exerted by that service are described.

1023

1024 • **Description.** Additional description and context for the element, along with explanatory notes and examples.  
 1025

1026 **5.1.5 Abstract Elements**

Element	Description	Fully Qualified Name (FQN)	Presence
<b>Action-status</b>	A status indication delivered by a service showing the success or failure of an action.	org.fipa.standard.service.action-status	Mandatory
<b>Agent</b>	A computational process that implements the autonomous, communicating functionality of an application.	org.fipa.standard.agent	Mandatory
<b>Agent-attribute</b>	A set of properties associated with an <b>agent</b> by inclusion in its <b>agent-directory-entry</b> .	org.fipa.standard.agent.agent-attribute	Optional
<b>Agent-communication-language</b>	A language with a precisely defined syntax semantics and pragmatics, which is the basis of communication between independently designed and developed <b>agents</b> .	org.fipa.standard.agent-communication-language	Mandatory
<b>Agent-directory-entry</b>	A composite entity containing the <b>name</b> , <b>agent-locator</b> , and <b>agent-attributes</b> of an <b>agent</b> .	org.fipa.standard.service.agent-directory-entry	Mandatory
<b>Agent-directory-service</b>	A <b>service</b> providing a shared information repository in which <b>agent-directory-entries</b> may be stored and queried	org.fipa.standard.service.agent-directory-service	Mandatory
<b>Agent-locator</b>	An <b>agent-locator</b> consists of the set of <b>transport-descriptions</b> used to communicate with an <b>agent</b> .	org.fipa.standard.service.message-transport-service.agent-locator	Mandatory
<b>Agent-name</b>	An opaque, non-forgeable token that uniquely identifies an <b>agent</b> .	org.fipa.standard.agent-name	Mandatory
<b>Content</b>	<b>Content</b> is that part of a <b>message</b> (communicative act) that represents the domain dependent component of the communication.	org.fipa.standard.message.content	Mandatory
<b>Content-language</b>	A language used to express the <b>content</b> of a communication between agents.	org.fipa.standard.message.content-language	Mandatory

<b>Encoding-representation</b>	A way of representing an abstract syntax in a particular concrete syntax. Examples of possible representations are XML, FIPA Strings, and serialized Java objects.	org.fipa.standard.encoding-service.encoding-representation	Mandatory
<b>Encoding-service</b>	A <b>service</b> that encodes a <b>message</b> to and from a <b>payload</b> .	org.fipa.standard.service.encoding-service	Mandatory
<b>Envelope</b>	That part of a <b>transport-message</b> containing information about how to send the message to the intended recipient(s). May also include additional information about the message encoding, encryption, etc.	org.fipa.standard.transport-message.envelope	Mandatory
<b>Explanation</b>	An encoding of the reason for a particular <b>action-status</b> .	org.fipa.standard.service.explanation	Optional
<b>Message</b>	A unit of communication between two agents. A <b>message</b> is expressed in an <b>agent-communication-language</b> , and encoded in an <b>encoding-representation</b> .	org.fipa.standard.message	Mandatory
<b>Message-transport-service</b>	A <b>service</b> that supports the sending and receiving of <b>transport-messages</b> between <b>agents</b> .	org.fipa.standard.service.message-transport-service	Mandatory
<b>Ontology</b>	A set of symbols together with an associated interpretation that may be shared by a community of <b>agents</b> or software. An ontology includes a vocabulary of symbols referring to objects in the subject domain, as well as symbols referring to relationships that may be evident in the domain.	org.fipa.standard.message.ontology	Optional
<b>Payload</b>	A <b>message</b> encoded in a manner suitable for inclusion in a <b>transport-message</b> .	org.fipa.standard.transport-message.payload	Mandatory
<b>Service</b>	A service provided for <b>agents</b> and other <b>services</b> .	org.fipa.standard.service	Mandatory
<b>Service-address</b>	A <b>service-type</b> specific string containing transport addressing information.	org.fipa.standard.service.service-address	Mandatory
<b>Service-attributes</b>	A set of properties associated with a <b>service</b> by inclusion in its <b>service-directory-entry</b> .	org.fipa.standard.service.service-attributes	Optional
<b>Service-directory-entry</b>	A composite entity containing the <b>service-nameid</b> , <b>service-locator</b> , and <b>service-type</b> of a <b>service</b> .	org.fipa.standard.service.service-directory-service.service-directory-entry	Mandatory
<b>Service-directory-service</b>	A directory service for registering and discovering <b>services</b> .	org.fipa.standard.service.service-directory-service	Mandatory
<b>Service-nameid</b>	A unique identifier of a particular <b>service</b> .	org.fipa.standard.service.service-idservice-name	Mandatory
<b>Service-location-description</b>	A <b>key-value-tuple</b> containing a <b>signature-type</b> a <b>service-signature</b> and <b>service-address</b> .	org.fipa.standard.service.service-location-description	Mandatory
<b>Service-locator</b>	A <b>service-locator</b> consists of the set of <b>service-location-descriptions</b> used to access a <b>service</b> .	org.fipa.standard.service.service-locator	Mandatory
<b>Service-root</b>	A set of <b>service-directory-entries</b> .	org.fipa.standard.service.service-root	Mandatory
<b>Service-signature</b>	A identifier that describes the binding signature for a <b>service</b> .	org.fipa.standard.service.service-type	Mandatory

<b>Service-type</b>	A <b>key-value tuple</b> describing the type of a <b>service</b> .	org.fipa.standard.service.service-type	Mandatory
<b>Signature-type</b>	A <b>key-value tuple</b> describing the type of <b>service-signature</b> .	org.fipa.standard.service.signature-type	
<b>Transport</b>	A <b>transport</b> is a particular data delivery service supported by a given <b>message-transport-service</b> .	org.fipa.standard.service.message-transport-service.transport	Mandatory
<b>Transport-description</b>	A <b>transport-description</b> is a self describing structure containing a <b>transport-type</b> , a <b>transport-specific-address</b> and zero or more <b>transport-specific-properties</b> .	org.fipa.standard.service.message-transport-service.transport-description	Mandatory
<b>Transport-message</b>	The object conveyed from <b>agent</b> to <b>agent</b> . It contains the <b>transport-description</b> for the sender and receiver or receivers, together with a <b>payload</b> containing the <b>message</b> .	org.fipa.standard.transport-message	Mandatory
<b>Transport-specific-address</b>	A transport address specific to a given <b>transport-type</b>	org.fipa.standard.service.message-transport-service.transport-specific-address	Mandatory
<b>Transport-specific-property</b>	A <b>transport-specific-property</b> is a property associated with a <b>transport-type</b> .	org.fipa.standard.service.message-transport-service.transport-specific-property	Optional
<b>Transport-type</b>	A <b>transport-type</b> describes the type of transport associated with a <b>transport-specific-address</b> .	org.fipa.standard.service.message-transport-service.transport-type	Mandatory

Table 2: Abstract Elements

## 5.2 Agent

### 5.2.1 Summary

An **agent** is a computational process that implements the autonomous, communicating functionality of an application. Typically, agents communicate using an **Agent Communication Language**. A concrete instantiation of **agent** is a mandatory element of every concrete instantiation of the abstract architecture.

### 5.2.2 Relationships to Other Elements

**Agent** has an **agent-name**

**Agent** may have **agent-attributes**

**Agent** has an **agent-locator**, which lists the **transport-descriptions** for that agent

**Agent** may be sent messages via a **transport-description**, using the **transport** corresponding to the **transport-description**

**Agent** may send a **transport-message** to one or more **agents**

**Agent** may register with one or more **agent-directory-services**

**Agent** may have an **agent-directory-entry**, which is registered with an **agent-directory-service**

**Agent** may modify its **agent-directory-entry** as registered by an **agent-directory-service**

**Agent** may **deletederegister** its **agent-directory-entry** from an **agent-directory-service**.

**Agent** may **querysearch** for an **agent-directory-entry** registered within an **agent-directory-service**

**Agent** is addressable by the mechanisms described in its **transport-descriptions** in its **agent-directory-entry**.

### 1050 5.2.3 Description

1051 In a concrete instantiation of the abstract architecture, an **agent** may be realized in a variety of ways, for example as a  
1052 Java™ component, a COM object, a self-contained Lisp program, or a TCL script. It may execute as a native process  
1053 on some physical computer under an operating system, or be supported by an interpreter such as a Java Virtual  
1054 Machine or a TCL system. The relationship between the **agent** and its computational context is specified by the agent  
1055 lifecycle. The abstract architecture does not address the lifecycle of agents as it is often handled differently in discrete  
1056 computational environments. Realizations of the abstract architecture *must* address these issues.  
1057

## 1058 5.3 Agent Attribute

### 1059 5.3.1 Summary

1060 An **agent-attribute** is one of a set of optional attributes that form part of the **agent-directory-entry** for an **agent**. They  
1061 are represented as **key-value-pairs** within the **key-value-tuple** that makes up the **agent-directory-entry**. The  
1062 purpose of the attributes is to allow searching for **agent-directory-entries** that match **agents** of interest. A concrete  
1063 instantiation of **agent-attribute** is an optional element of concrete instantiations of the abstract architecture.  
1064

### 1065 5.3.2 Relationships to Other Elements

1066 An **agent-directory-entry** may have zero or more **agent-attributes**  
1067 An **agent-attribute** describes aspects of an **agent**  
1068

### 1069 5.3.3 Description

1070 When an **agent** registers an **agent-directory-entry**, the **agent-directory-entry** may optionally contain **key-value-**  
1071 **pairs** that offer additional description of the **agent**. The values might include information about costs of using the  
1072 **agent** or **service**, features available, **ontologies** understood, names that the service is commonly known by, or any  
1073 other data that agents deem useful. This information can then be used to enhance search criteria exerted by **agents**  
1074 on the **agent-directory-service**.  
1075

1076 In practice, when defining realizations of this abstract architecture, domain specific specifications should exist  
1077 describing the **agent-attributes** to be used. This eases requirements for interoperation.  
1078

## 1079 5.4 Agent Communication Language

### 1080 5.4.1 Summary

1081 An **agent-communication-language** (ACL) is a language in which communicative acts can be expressed and hence  
1082 **messages** constructed. A concrete instantiation of **agent-communication-language** is a mandatory element of every  
1083 concrete instantiation of the abstract architecture.  
1084

### 1085 5.4.2 Relationships to Other Elements

1086 **Message** is written in an **agent-communication-language**

### 1087 5.4.3 Description

1088 FIPA ACL is described in detail in [FIPA00061] and FIPA communicative acts in [FIPA00037].  
1089

## 5.5 Agent Directory Entry

### 5.5.1 Summary

An **agent-directory-entry** is a **key-value tuple** consisting of the **agent-name**, an **agent-locator**, and zero or more **agent-attributes**. An **agent-directory-entry** refers to an **agent**; in some implementations this agent will provide a **service**. A concrete instantiation of **agent-directory-entry** is a mandatory element of every concrete instantiation of the abstract architecture.

### 5.5.2 Relationships to Other Elements

**Agent-directory-entry** contains the **agent-name** of the **agent** to which it refers

**Agent-directory-entry** contains one **agent-locator** of the **agent** to which it refers. The **agent-locator** contains one or more **transport-descriptions**

**Agent-directory-entry** is managed by and available from an **agent-directory-service**

**Agent-directory-entry** may contain **agent-attributes**

### 5.5.3 Description

Different realizations that use a common **agent-directory-service**, are strongly encouraged to adopt a common schema for storing **agent-directory-entries**. (This in turn implies the use of a common representation for **agent-locators**, **transport-descriptions**, **agent-names**, and so forth.)

**Agents** are not required to publish an **agent-directory-entry**. It is possible for agents to communicate with agents that have provided a **transport-description** through a private mechanism. For example, an agent involved in a negotiation may receive a **transport-description** directly from the party with which it is negotiating. This falls outside the scope of the using the **agent-directory-services** mechanisms.

## 5.6 Agent Directory Service

### 5.6.1 Summary

An **agent-directory-service** is a shared information repository in which **agents** may publish their **agent-directory-entries** and in which they may search for **agent-directory-entries** of interest. A concrete instantiation of **agent-directory-service** is a mandatory element of every concrete instantiation of the abstract architecture.

### 5.6.2 Relationships to Other Elements

**Agent** may register its **agent-directory-entry** with an **agent-directory-service**

**Agent** may modify its **agent-directory-entry** as registered by an **agent-directory-service**

**Agent** may ~~delete~~**deregister** its **agent-directory-entry** from an **agent-directory-service**

**Agent** may search for an **agent-directory-entry** registered within an **agent-directory-service**

An **agent-directory-service** must accept valid, authorized requests to register, de-register, ~~delete~~, modify and identify agent descriptions

An **agent-directory-service** must accept valid, authorized requests for searching

### 5.6.3 Actions

An **agent-directory-service** supports the following actions.

#### 5.6.3.1 Register

An **agent** may **register** an **agent-directory-entry** with an **agent-directory-service**. The semantics of this action are as follows:

The **agent** provides an **agent-directory-entry** that is to be registered. In initiating the action, the **agent** may control the scope of the action. Different reifications may handle this in different ways. The action may be addressed to a particular instance of an **agent-directory-service**, or the action may be qualified with some kind of scope parameter.

If the action is successful, the **agent-directory-service** will return an **action-status** indicating success. Following a successful **register**, the **agent-directory-service** will support legal **modify**, **deletere**, and **querysearch** actions with respect to the registered **agent-directory-entry**.

If the action is unsuccessful, the **agent-directory-service** will return an **action-status** indicating failure, together with an **explanation**. The range of possible explanations is, in general, specific to a particular reification. However a conforming reification must, where appropriate, distinguish between the following explanations:

- *Duplicate*. The new entry "clashed" with some existing **agent-directory-entry**. Normally this would only occur if an existing **agent-directory-entry** had the same **agent-name**, but specific reifications may enforce additional requirements.
- *Access*. The **agent** making the request is not authorized to perform the specified action.
- *Invalid*. The **agent-directory-entry** is invalid in some way.

#### 5.6.3.2 Modify

An **agent** may **modify** an **agent-directory-entry** that has been registered with an **agent-directory-service**. The semantics of this action are as follows:

The **agent** provides an **agent-directory-entry** which contains the same **agent-name** as the entry to be modified. In initiating the action, the **agent** may control the scope of the action. Different reifications may handle this in different ways. The action may be addressed to a particular instance of an **agent-directory-service**, or the action may be qualified with some kind of scope parameter.

The **agent-directory-service** verifies that the argument is a valid **agent-directory-entry**. It then searches for a registered **agent-directory-entry** with the same **agent-name**. If it does not find one, the action fails and an **explanation** provided. Otherwise it modifies the existing **agent-directory-entry** by examining each **key-value pair** in new **agent-directory-entry**. If the **value** is non-null, the **pair** is added to the new entry, replacing any existing **pair** with the same **key**. If the **value** is null, any existing **pair** with the same **key** is removed from the entry.

If the action is successful, the **agent-directory-service** will return an **action-status** indicating success, together with an **agent-directory-entry** corresponding to the new contents of the registered entry. Following a successful **register**, the **agent-directory-service** will support legal **modify**, **deletere**, and **querysearch** actions with respect to the modified **agent-directory-entry**.

If the action is unsuccessful, the **agent-directory-service** will return an **action-status** indicating failure, together with an **explanation**. The range of possible explanations is, in general, specific to a particular reification. However a conforming reification must, where appropriate, distinguish between the following explanations:

- *Not-found*. The new entry did not match any existing **agent-directory-entry**. This would only occur if no existing **agent-directory-entry** had the same **agent-name**.
- *Access*. The **agent** making the request is not authorized to perform the specified action.
- *Invalid*. The new **agent-directory-entry** is invalid in some way.

#### 5.6.3.3 De**registerlete**

An **agent** may **deletere** an **agent-directory-entry** from an **agent-directory-service**. The semantics of this action are as follows:

The **agent** provides an **agent-directory-entry** which has the same **agent-name** as that which is to be **deleted/registered**. (The rest of the **agent-directory-entry** is not significant.) In initiating the action, the **agent** may control the scope of the action. Different reifications may handle this in different ways. The action may be addressed to a particular instance of an **agent-directory-service**, or the action may be qualified with some kind of scope parameter.

If the action is successful, the **agent-directory-service** will return an **action-status** indicating success. Following a successful **deleted/registered**, the **agent-directory-service** will no longer support **modify**, **deleted/registered**, and **querysearch** actions with respect to the registered **agent-directory-entry**.

If the action is unsuccessful, the **agent-directory-service** will return an **action-status** indicating failure, together with an **explanation**. The range of possible explanations is, in general, specific to a particular reification. However a conforming reification must, where appropriate, distinguish between the following explanations:

- *Not-found*. The new entry did not match any existing **agent-directory-entry**. This would only occur if no existing **agent-directory-entry** had the same **agent-name**.
- *Access*. The **agent** making the request is not authorized to perform the specified action.
- *Invalid*. The **agent-directory-entry** is invalid in some way.

#### 5.6.3.4 QuerySearch

An **agent** may **querysearch** an **agent-directory-service** to locate **agent-directory-entries** of interest. The semantics of this action are as follows:

The **agent** provides an **agent-directory-entry** that is to be treated as a search pattern. In initiating the action, the **agent** may control the scope of the action. Different reifications may handle this in different ways. The action may be addressed to a particular instance of an **agent-directory-service**, or the action may be qualified with some kind of scope parameter.

The directory service verifies that the argument is a valid **agent-directory-entry**. It then searches for registered **agent-directory-entries** that satisfy the search criteria. A registered entry satisfies the search criteria if there is a match between each **key-value pair** in the submitted entry. The semantics of "matching" are likely to be reification-dependent; at a minimum, there should be support for matching on the *same* value and on *any* value.

If the action is successful, the **agent-directory-service** will return an **action-status** indicating success, together with a set of **agent-directory-entries** that satisfy the search pattern. The mechanism by which multiple entries are returned, and whether or not an agent may limit or terminate the delivery of results, is not defined in the abstract architecture and is therefore reification dependent.

If the action is unsuccessful, the **agent-directory-service** will return an **action-status** indicating failure, together with an **explanation**. The range of possible explanations is, in general, specific to a particular reification. However a conforming reification must, where appropriate, distinguish between the following explanations:

- *Not-found*. The search pattern did not match any existing **agent-directory-entry**.
- *Access*. The **agent** making the request is not authorized to perform the specified action.
- *Invalid*. The **agent-directory-entry** is invalid in some way.

#### 5.6.4 Description

An **agent-directory-service** may be implemented in a variety of ways, using a general-purpose scheme such as X.500 or some private agent-specific mechanism. Typically an **agent-directory-service** uses some hierarchical or

1243 federated scheme to support scalability. A concrete implementation may support such mechanisms automatically, or  
 1244 may require each **agent** to manage its own directory usage.

1245  
 1246 Different realizations that are based on the same underlying mechanism are strongly encouraged to adopt a common  
 1247 schema for storing **agent-directory-entries**. This in turn implies the use of a common representation for **names**,  
 1248 **locations**, and so forth. For example, considering multiple implementations of directory services in LDAP, it would be  
 1249 useful for all of the implementations to interoperate because they are using the same schemas. Similarly, if there were  
 1250 multiple implementations in NIS, they would need the same NIS data representation to interoperate.

1251  
 1252 The **agent-directory-service** described here does not have the full flexibility found in the *directory-facilitator* (see  
 1253 [FIPA00023]), of existing FIPA specifications. In practice, implementing the search capabilities of the existing *directory-*  
 1254 *facilitator* is not feasible with most directory systems, that is, LDAP, X.500 and NIS. There seems to be a need for a  
 1255 Lookup Service, which is here called the **agent-directory-service**, which allows an agent to identify and get the  
 1256 **transport-description** for another agent, as well as a more complex search system, which can resolve complex  
 1257 searches. The former system, which supports a single level of search on attributes, is the **agent-directory-service**.  
 1258 The latter might be implemented as a broker, and might be implemented in systems that allow for arbitrary complexity  
 1259 and nesting such as Prolog or LISP. This division of functionality reflects the experience of many implementations,  
 1260 where there is a "quick" lookup service and a more robust, but slower complex search service.

## 1262 5.7 Agent Locator

### 1263 5.7.1 Summary

1264 An **agent-locator** consists of the set of **transport-descriptions**, which can be used to communicate with an **agent**. An  
 1265 **agent-locator** may be used by a **message-transport-service** to select a **transport** for communicating with the **agent**,  
 1266 such as an agent or a **service**. **Agent-locators** can also contain references to software interfaces. This can be used  
 1267 when a **service** can be accessed programmatically, rather than via a messaging model. A concrete instantiation of  
 1268 **agent-locator** is a mandatory element of every concrete instantiation of the abstract architecture.

### 1270 5.7.2 Relationships to Other Elements

1271 **Agent-locator** is a member of **agent-directory-entry**, which is registered with an **agent-directory-service**

1272 **Agent-locator** contains one or more **transport-descriptions**

1273 **Agent-locator** is used by **message-transport-service** to select a **transport**

### 1275 5.7.3 Description

1276 The **agent-locator** serves as a basic building block for managing address and transport resolution. An **agent-locator**  
 1277 includes all of the **transport-descriptions** that may be used to contact the related **agent** or **service**.

## 1279 5.8 Agent Name

### 1280 5.8.1 Summary

1281 An **agent-name** is a means to identify an **agent** to other **agents** and **services**. It is expressed as a **key-value-pair**, is  
 1282 unchanging (that is, it is immutable), and unique under normal circumstances of operation. A concrete instantiation of  
 1283 **agent-name** is a mandatory element of every concrete instantiation of the abstract architecture.

### 1285 5.8.2 Relationships to Other Elements

1286 **Agent** has one **agent-name**

1287 **Message** must contain the **agent-names** of the sending and receiving **agents**

1288 **Agent-directory-entry** must contain the **agent-name** of the **agent** to which it refers

### 5.8.3 Description

An **agent-name** is an identifier (e.g., a GUID, Globally Unique Identifier) that is associated with the **agent** at creation time or initial registration. Name issuing should occur in a way that tends to ensure global uniqueness. This may be achieved, for example, through employing an algorithm that generates the name with a sufficient degree of stochastic complexity such as to induce a vanishingly small chance of a name collision.

The **agent-name** will typically be issued by another entity or service. Once issued, the unique identifier should not be dependent upon the continued existence of the third party that issued it. Ideally through, there will be some mechanism available that is capable of verifying name authenticity.

Beyond this durable relationship with the **agent** it denotes, the **agent-name** should have no semantics. It should not encode any actual properties of the agent itself, nor should it disclose related information such as agent **transport-description** or **location**. It should also not be used as a form of authentication of the agent. Authentication services must rely on the combination of a unique identifier plus additional information (for example, a certificate that makes the name tamper-proof and verifies its authenticity through a trusted third party).

A useful role of an **agent-name** is to support the use of BDI (belief/desire/intention) models within a multi-agent system. The **agent-name** can be used to correlate propositional attitudes with the particular **agents** that are believed to hold those attitudes.

**Agents** may also have "well-known" or "common" or "social" names, or "nicknames", or aliases by which they are popularly known. These names are often used to commonly identify an agent. For example, within an agent system, there may be a broker service for finding "air-fare" agents. The convention within this system is that this agent is nicknamed "Air-fare broker". In practice, this is implemented as an **agent-attribute**. The attribute could have the key "Nickname" with the value "Air-fare broker". However, the actual name of the agent providing the function is unique, to maintain the ability to distinguish between an agent providing that function in one cluster of agents, and another agent providing the same function in a different cluster of agents.

## 5.9 Content

### 5.9.1 Summary

**Content** is that part of a **message** (where a message is a communicative act) that represents the component of the communication that refers to a domain or topic area. **Content** is expressed using **content-languages**. Expressions contained within the content, or the entire content expression itself, can be put into context by one or more **ontologies**. A concrete instantiation of **content** is a mandatory element of every concrete instantiation of the abstract architecture.

### 5.9.2 Relationships to Other Elements

**Content** is expressed in a **content-language**

**Content** may reference one or more ontologies referenced in the **ontology** attribute of a **message**

**Content** is part of a **message**

### 5.9.3 Description

The **content** of a **message** is the propositional content of a speech act. It does not refer to everything within the message, including delimiters, as it does with some languages, but rather the domain specific component only.

## 5.10 Content Language

### 5.10.1 Summary

A **content-language** is a language used to express the **content** of a communication between agents. FIPA allows considerable flexibility in the choice, form and encoding of a content language. However, content languages are

1338 required to be able to represent propositions, actions and terms (names of individual entities) if they are to make full  
 1339 use of the standard FIPA performatives. A concrete instantiation of **content-language** is a mandatory element of every  
 1340 concrete instantiation of the abstract architecture.  
 1341

### 1342 5.10.2 Relationships to Other Elements

1343 **Content** is expressed in a **content-language**  
 1344 **FIPA-SL** is an example of a **content-language**  
 1345 **FIPA-RDF** is an example of a **content-language**  
 1346 **FIPA-KIF** is an example of a **content-language**  
 1347 **FIPA-CCL** is an example of a **content-language**  
 1348

### 1349 5.10.3 Description

1350 The FIPA content language library is described in detail in [FIPA00007].  
 1351

## 1352 5.11 Encoding Representation

### 1353 5.11.1 Summary

1354 An **encoding-representation** is a way of representing a **message** in a particular transport encoding. Examples of  
 1355 possible representations are XML, Bit-efficient encoding and serialized Java objects. Typically an **encoding-**  
 1356 **representation** is applied to the **payload** component of a **transport-message** to prepare it for transmission. A  
 1357 concrete instantiation of **encoding-representation** is a mandatory element of every concrete instantiation of the  
 1358 abstract architecture.  
 1359

### 1360 5.11.2 Relationships to Other Elements

1361 **Payload** and the **message** and **content** contained within is encoded according to an **encoding-representation**  
 1362 **Encoding-representation** is used by an **encoding-service**

### 1363 5.11.3 Description

1364 The way in which a message is encoded depends on the concrete architecture. If a particular architecture supports  
 1365 only one form of encoding, no additional information is required. If multiple forms of encoding are supported, messages  
 1366 may be made self-describing using techniques such as format tags, object introspection, and XML DTD references.  
 1367

## 1368 5.12 Encoding Service

### 1369 5.12.1 Summary

1370 An **encoding-service** is a **service**. It provides the facility to encode a **message** or **content** into an **encoding-**  
 1371 **representation** for use as a **transport-message payload**. This procedure must also function in reverse for decoding  
 1372 **transport-messages**. A concrete instantiation of **encoding-service** is a mandatory element of every concrete  
 1373 instantiation of the abstract architecture.  
 1374

### 1375 5.12.2 Relationships to Other Elements

1376 **Encoding-service** converts a message into an **encoding-representation**  
 1377 **Encoding-service** converts an **encoding-representation** into a **message**  
 1378 **Encoding-service** can encode a **message** and message **content** as a **payload**  
 1379 **Encoding-service** can decode a **payload** into a **message**  
 1380 **Encoding-service** is a **service**  
 1381

### 5.12.3 Actions

An **encoding-service** supports the following actions.

#### 5.12.3.1 Transform Encoding/Decoding

An **agent** uses an **encoding-service** to convert a **message** to a **payload** and vice versa. That is, between **message** representation and a particular **encoding-representation**. It does this by invoking the **transform-encoding** action of the **encoding-service**. The semantics of this action are as follows:

To encode a message, the **agent** provides the **message** to the **encoding-service**, along with the type of encoding to be used. The encodings offered by the service may be queried using the **query-available-encodings** action described below. Encoding is context sensitive to ensure that appropriate **encoding-representations** are applied to specific message components. I.e. a **message** may be encoded in XML representation, but the **payload** that contains that **message** must be encoded for the transport to be used.

To decode a message, the encoded **payload** component of a **transport-message** is handed off to the **encoding-service** which decodes it into the **message**.

If the action is successful, the **encoding-service** will return an **action-status** indicating success, together with the encoded message component.

If the action is unsuccessful, the **encoding-service** will return an **action-status** indicating failure, together with an **explanation**. The range of possible explanations is, in general, specific to a particular reification. However a conforming reification must, where appropriate, distinguish between the following explanations:

- *Access*. The **agent** making the request is not authorized to perform the specified action.
- *Invalid Message*. The **message** to be encoded is invalid in some way.
- *Invalid Payload*. The **payload** to be decoded is invalid in some way.
- *Invalid Encoding*. The **encoding-representation** selected is unavailable.

#### 5.12.3.2 Query Encoding Representation

An **agent** may query the **encoding-service** to resolve the **encoding-representation** with which the supplied **payload** has been encoded. It does this by invoking the **query-encoding-representation** action of the **encoding-transform-service**.

If the action is successful, the **encoding-service** will return an **action-status** indicating success. Additionally, the **encoding-representation** identity is returned.

If the action is unsuccessful, the **encoding-service** will return an **action-status** indicating failure, together with an **explanation**. The range of possible explanations is, in general, specific to a particular reification. However a conforming reification must, where appropriate, distinguish between the following explanations:

- *Access*. The **agent** making the request is not authorized to perform the specified action.
- *Invalid*. The encoded **payload** is invalid in some way.
- *Unidentifiable*. The **encoding-representation** is unidentifiable by the **encoding-service**.

#### 5.12.3.3 Query Available Encodings

An **agent** may query the **encoding-service** to provide a list of all **encoding-representations** known by the service. It does this by invoking the **query-available-encodings** action of the **encoding-service**.

1435

1436 If the action is successful, the **encoding-service** will return an **action-status** indicating success. Additionally, the  
 1437 available **encoding-representations** are supplied.

1438

1439 If the action is unsuccessful, the **encoding-service** will return an **action-status** indicating failure, together with an  
 1440 **explanation**. The range of possible explanations is, in general, specific to a particular reification. However a  
 1441 conforming reification must, where appropriate, distinguish between the following explanations:

1442

- Access. The **agent** making the request is not authorized to perform the specified action.

1444

#### 1445 5.12.4 Description

1446 A concrete specification must realize a reification of the **encoding-service** in order that **agents** can encode and  
 1447 decode **encoding-representations** from and into a **message** representation, respectively. Every individual **encoding-**  
 1448 **representation** will require a specific codec for transforming to and from any **message** and **content** representation.

1449

### 1450 5.13 Envelope

#### 1451 5.13.1 Summary

1452 An **envelope** is a **key-value tuple** that contains message delivery and encoding information. It is included in the  
 1453 **transport-message**, and includes elements such as the sender and receiver(s) **transport-descriptions**. It also  
 1454 contains the **encoding-representation** for the **message** and optionally, other message information such as validation  
 1455 and security data, or additional routing data. A concrete instantiation of **envelope** is a mandatory element of every  
 1456 concrete instantiation of the abstract architecture.

1457

#### 1458 5.13.2 Relationship to Other Elements

1459 **Envelope** contains **transport-descriptions**

1460 **Envelope** optionally contains validity data (such as security keys for message validation)

1461 **Envelope** optionally contains security data (such as security keys for message encryption or decryption)

1462 **Envelope** optionally contains routing data

1463 **Envelope** contains an **encoding-representation** for the **payload** being transported

1464 **Envelope** is contained in **transport-message**

1465

#### 1466 5.13.3 Description

1467 In the realization of the envelope data, the realization can specify envelope elements that are useful in the particular  
 1468 realization. These can include specialized routing data, security related data, or other data that can assist in the proper  
 1469 handling of the encoded **message**.

1470

### 1471 5.14 Explanation

#### 1472 5.14.1 Summary

1473 An encoding of the reason for a particular **action-status**. When an action exerted by a service leads to a failure  
 1474 response, the **explanation** is an optional descriptor giving the reason why the particular action failed. A concrete  
 1475 instantiation of **explanation** is an optional element of every concrete instantiation of the abstract architecture.

1476

#### 1477 5.14.2 Relationship to Other Elements

1478 **Explanation** qualifies an **action-status**.

1479

### 1480 5.14.3 Description

1481 In terms of the three explicit services described by the abstract architecture, the **agent-directory-service**, **service-**  
1482 **directory-service** and **message-transport-service**, the relevant action **explanations** are listed in the appropriate  
1483 element subsections.  
1484

## 1485 5.15 Message

### 1486 5.15.1 Summary

1487 A **message** is an individual unit of communication between two or more **agents**. A **message** logically arises from and  
1488 programmatically corresponds to a communicative act, in the sense that a **message** encodes the communicative act.  
1489 Communicative acts can be recursively composed, so while the outermost act is directly encoded by the **message**,  
1490 taken as a whole a given **message** may represent multiple individual communicative acts. This is then encoded using  
1491 an **encoding-representation** and transmitted between **agents** over a **transport**. A **message** includes an indication of  
1492 the type of communicative act (for example, INFORM, REQUEST), the **agent-names** of the sender and receiver  
1493 **agents**, the **ontology** or **ontologies** to be used in interpreting the **content**, and the **content** of the **message** itself. A  
1494 **message** does not include any transport or addressing information. It is transmitted from sender to receiver(s) by  
1495 being encoded as the **payload** of a **transport-message**, which includes this information. A concrete instantiation of  
1496 **message** is a mandatory element of every concrete instantiation of the abstract architecture.  
1497

### 1498 5.15.2 Relationships to other elements

1499 **Message** is written in an **agent-communication-language**

1500 **Message** contains **content**

1501 **Message** has an **ontology** attribute

1502 **Message** includes an **agent-name** corresponding to the sender of the message

1503 **Message** includes one or more **agent-name** corresponding to the receiver or receivers of the message

1504 **Message** is sent by an **agent**

1505 **Message** is received by one or more **agents**

1506 **Message** is transmitted as the **payload** of a **transport-message**

1507 **Message** is transformed to/from a **payload** by an **encoding-service**  
1508

### 1509 5.15.3 Description

1510 The FIPA communicative acts library is described in detail in [FIPA00037].  
1511

## 1512 5.16 Message Transport Service

### 1513 5.16.1 Summary

1514 A **message-transport-service** is a **service**. It supports the sending and receiving of **transport-messages** between  
1515 **agents**. A concrete instantiation of **message-transport-service** is a mandatory element of every concrete instantiation  
1516 of the abstract architecture.  
1517

### 1518 5.16.2 Relationships to Other Elements

1519 **Message-transport-service** may be invoked to send a **transport-message** to an **agent**

1520 **Message-transport-service** selects a **transport** based on the recipient's **transport-description**

1521 **Message-transport-service** is a **service**  
1522

### 1523 5.16.3 Actions

1524 A **message-transport-service** supports the following actions.

1525

## 1526 5.16.3.1 Bind Transport

1527 An **agent** may form a contract with the **message-transport-service** to send and receive messages using a particular  
 1528 **transport**. It does this by invoking the **bind-transport** action of the **message-transport-service**. The semantics of  
 1529 this action are as follows:

1530

1531 The **agent** provides a **transport-description** corresponding to the **transport** to be used. (In initiating the action, the  
 1532 **agent** may control the scope of the action. Different reifications may handle this in different ways. The action may be  
 1533 addressed to a particular instance of a **agent-directory-service**, or the action may be qualified with some kind of  
 1534 scope parameter.) Some or all of the elements of the **transport-description** may be missing, in which case the  
 1535 **transport-service** may supply appropriate values. The **transport-service** attempts to create a usable transport-end-  
 1536 point for the chosen **transport-type**, and constructs a **transport-specific-address** corresponding to this end-point.

1537

1538 If the action is successful, the **message-transport-service** will return an **action-status** indicating such, together with a  
 1539 **transport-description** that has been completely filled in and is usable for message transport. The agent may use this  
 1540 **transport-description** as part of its **agent-description**, and in constructing a **transport-message**.

1541

1542 Following a successful **bind-transport**, the **message-transport-service** will attempt to deliver any messages received  
 1543 over the transport end-point to the **agent**.

1544

1545 If the action is unsuccessful, the **message-transport-service** will return an **action-status** indicating failure, together  
 1546 with an **explanation**. The range of possible explanations is, in general, specific to a particular reification. However a  
 1547 conforming reification must, where appropriate, distinguish between the following explanations:

1548

- 1549 • *Access*. The **agent** making the request is not authorized to perform the specified action.

1550

- 1551 • *Invalid*. The **transport-description** is invalid in some way.

1552

## 1553 5.16.3.2 Unbind Transport

1554 An **agent** may terminate a contract with the **message-transport-service** to send and receive messages using a  
 1555 particular **transport**. It does this by invoking the **unbind-transport** action of the **message-transport-service**. The  
 1556 semantics of this action are as follows:

1557

1558 The **agent** provides a **transport-description** returned by a previous **bind-transport** action. (In initiating the action, the  
 1559 **agent** may control the scope of the action. Different reifications may handle this in different ways. The action may be  
 1560 addressed to a particular instance of a **agent-directory-service**, or the action may be qualified with some kind of  
 1561 scope parameter.) The **transport-service** identifies the corresponding transport-end-point and releases all transport  
 1562 related resources.

1563

1564 If the action is successful, the **message-transport-service** will return an **action-status** indicating success.  
 1565 Additionally, the **message-transport-service** will no longer attempt to deliver any messages to the **agents** associated  
 1566 with the defunct transport binding.

1567

1568 If the action is unsuccessful, the **message-transport-service** will return an **action-status** indicating failure, together  
 1569 with an **explanation**. The range of possible explanations is, in general, specific to a particular reification. However a  
 1570 conforming reification must, where appropriate, distinguish between the following explanations:

1571

- 1572 • *Not-found*. The **transport-description** does not correspond to a bound **transport**.

1573

- 1574 • *Access*. The **agent** making the request is not authorized to perform the specified action.

1575

- 1576 • *Invalid*. The **transport-description** is invalid in some way.

1577

### 5.16.3.3 Send Message

An **agent** may send a **transport-message** to another agent by invoking the **send-message** action of a **message-transport-service**. The semantics of this action are as follows:

The **agent** provides a **transport-message** to be sent. The **message-transport-service** examines the **envelope** of the message to determine how it should be handled.

If the action is successful, the **message-transport-service** will return an **action-status** indicating success. Following a successful **send-message**, the **message-transport-service** will make attempt to deliver the message to the recipient. However the successful completion of the **send-message** action should not be interpreted as indicating that delivery has been achieved.

If the action is unsuccessful, the **message-transport-service** will return an **action-status** indicating failure, together with an **explanation**. The range of possible explanations is, in general, specific to a particular reification. However a conforming reification must, where appropriate, distinguish between the following explanations:

- *Access*. The **agent** making the request is not authorized to perform the specified action.
- *Invalid*. The **transport-message** is invalid in some way.

### 5.16.3.4 Deliver Message

A **message-transport-service** may deliver a **transport-message** to an **agent** by invoking the **deliver-message** action of the **agent**. The semantics of this action are identical to those given for the **bind-transport** action.

## 5.16.4 Description

A concrete specification need not realize the notion of **message-transport-service** so long as the basic service provisions are satisfied. In the case of a concrete specification based on a single **transport**, the agent may use native operating system services or other mechanisms to achieve this service.

## 5.17 Ontology

### 5.17.1 Summary

An **Ontology** provides a vocabulary for representing and communicating knowledge about some topic and a set of relationships and properties that hold for the entities denoted by that vocabulary. A concrete instantiation of **ontology** is an optional element of concrete instantiations of the abstract architecture.

### 5.17.2 Relationships to Other Elements

**Message** has an **ontology** attribute that can contain references to one or more ontologies

**Content** is expressed in the context of one or more ontologies using the **ontology** message attribute

### 5.17.3 Description

An **ontology** is a set of symbols together with an associated interpretation that may be shared by a community of **agents** or **services**. An **ontology** includes a vocabulary of symbols referring to objects and relationships in the subject domain. An **ontology** also typically includes a set of logical statements expressing the constraints existing in the domain and restricting the interpretation of the vocabulary.

**Ontologies** must be nameable, discoverable and manageable.

## 1625 5.18 Payload

### 1626 5.18.1 Summary

1627 A **payload** is a **message** encoded in a manner suitable for inclusion in a **transport-message**. A concrete instantiation  
1628 of **payload** is a mandatory element of every concrete instantiation of the abstract architecture.  
1629

### 1630 5.18.2 Relationships to Other Elements

1631 **Payload** is an encoded **message**

1632 **Transport-message** contains a **payload**

1633 **Payload** is encoded according to an **encoding-representation**  
1634

### 1635 5.18.3 Description

1636 See Section 5.33.2, *Relationships to Other Elements* which describes the **transport-message** element.  
1637

## 1638 5.19 Service

### 1639 5.19.1 Summary

1640 A **service** is a functional coherent set of mechanisms that support the operation of **agents**, and other **services**. These  
1641 are services used in the provisioning of *agent environments* and may be used as the basis for interoperation. A  
1642 concrete instantiation of **service** is a mandatory element of every concrete instantiation of the abstract architecture.  
1643

1644 Note: A service in this specification should not be confused with the service or services provided by agents  
1645 implemented within instantiations of the architecture.  
1646

### 1647 5.19.2 Relationships to Other Elements

1648 **Service** has a public set of behaviours and actions

1649 **Service** has a service description

1650 **Service** can be accessed by **agents**

1651 **Agent-directory-service** is an instance of **service**, and is mandatory

1652 **Message-transport-service** is an instance of **service**, and is mandatory

1653 **Service-directory-service** is an instance of **service**, and is mandatory

1654 A **service** has a **service-type**, a **service-id****service-name**, a **service-locator**

1655 A **service** can have a **service-directory-entry** in a **service-directory-service** containing the **service-id****service-**  
1656 **name**, **service-type** and **service-locator**  
1657

### 1658 5.19.3 Description

1659 FIPA will administer the name space of **services** according to the description given in Section 5.1.2. This is part of the  
1660 concrete realization process. Having a clear naming scheme for the **services** will allow for optimised implementation  
1661 and management of **services**.  
1662

## 1663 5.20 Service Address

### 1664 5.20.1 Summary

1665 A **service-type** specific string that indicates how to bind to a particular **service**. A concrete instantiation of **service-**  
1666 **address** is a mandatory element of every concrete instantiation of the abstract architecture.

## 1667 5.20.2 Relationships to Other Elements

1668 **Service-address** provides an address of a **service** that can be bound to by an **agent** or **service**

1669 **Services-locators** contain one or more **service-addresses**

1670 A **service-address** is qualified by a **signature-type**

## 1672 5.20.3 Description

1673 The **service address** is a **service-type** specific string that indicates how to bind to a **service**. The precise means by  
 1674 which this binding is made is implementation and **service-type** specific; for example a **transport-service** that is bound  
 1675 via RMI objects may give an RMI address of the Java object to bind to and thereby access the **transport-service**.  
 1676 Alternatively, an **agent-directory-service** that is accessed via a TCP/IP socket may give a string containing the  
 1677 hostname and port number.  
 1678

## 1679 5.21 Service Attributes

### 1680 5.21.1 Summary

1681 **Service-attributes** are optional attributes that are part of the **service-directory-entry** for a **service**. They are  
 1682 represented as **key-value-pairs** within the **key-value-tuple** that makes up the **service-directory-entry**. The purpose  
 1683 of the attributes is to allow searching for **service-directory-entries** that match **services** of interest. A concrete  
 1684 instantiation of **service-attributes** is an optional element of concrete instantiations of the abstract architecture.  
 1685

### 1686 5.21.2 Relationships to Other Elements

1687 A **service-directory-entry** may have zero or more **service-attributes**

1688 **Service-attributes** describe aspects of a **service**

### 1690 5.21.3 Description

1691 When a **service** registers a **service-directory-entry**, the **service-directory-entry** may optionally contain **key-value-**  
 1692 **pairs** that offer additional description of the **service**. The values might include information about costs of using the  
 1693 **service**, features available, **ontologies** understood, names that the **service** is commonly known by, or any other  
 1694 relevant data. This information can then be used to enhance the search criteria by which **services** are discovered in  
 1695 the **service-directory-service**.  
 1696

1697 In practice, when defining realizations of this abstract architecture, domain specific specifications should exist  
 1698 describing the **service-attributes** to be used. This eases requirements for interoperation.  
 1699

## 1700 5.22 Service Directory Entry

### 1701 5.22.1 Summary

1702 A **service-directory-entry** is a **key-value-tuple** consisting of a **service-id****service-name**, **service-type**, **service-**  
 1703 **locator** and zero or more **service-attributes**. A concrete instantiation of **service-directory-entry** is a mandatory  
 1704 element of every concrete instantiation of the abstract architecture.  
 1705

### 1706 5.22.2 Relationships to Other Elements

1707 **Service-directory-entry** contains the **service-id****service-name** of the **service** to which it refers

1708 **Service-directory-entry** contains the **service-type** of the **service** to which it refers

1709 **Service-directory-entry** contains a **service-locator** of the **service** to which it refers

1710 **Service-directory-entry** may contain zero or more **service-attributes**

1711 **Service-directory-entry** is managed by and available from a **service-directory-service**

1712 **Services** are not required to publish a **service-directory-entry**

### 1714 5.22.3 Description

1715 A **service-directory-entry** is used to describe the identity, type, signature and address of a **service**, which is  
 1716 accessed via programmatic means. A **service-directory-entry** also contains zero or more attribute value pairs, which  
 1717 are used to distinguish on instance of a service from another. **Services** are registered to a **service-directory-service**  
 1718 by adding a **service-directory-entry** to the directory.

1719  
 1720 Different realizations that use a common **service-directory-service**, are strongly encouraged to adopt a common  
 1721 schema for storing **service-directory-entries**.

## 1723 5.23 Services Directory Service

### 1724 5.23.1 Summary

1725 The **service-directory-service** is used to register and locate **services** within the FIPA infrastructure. Services  
 1726 include, but are not limited to: **message-transport-services**, **agent-directory-services**, gateway services, and  
 1727 message buffering services (note that the latter two services are not mandated by this specification). A **service-**  
 1728 **directory-service** is also used to store the **service** descriptions of application oriented services, such as commercial  
 1729 and business oriented services. A concrete instantiation of **service-directory-service** is a mandatory element of every  
 1730 concrete instantiation of the abstract architecture.

1731  
 1732 Note: Agents are not expected to register services in the **services-directory-service** which are not being used in  
 1733 explicit provision of services for the platform. In addition, it would be expected that most services would not be register  
 1734 by agents.

### 1735 5.23.2 Relationships to Other Elements

1736 **Service-directory-services** provides a directory of **service-directory-entries**

1737 **Services** may be registered within the **service-directory-service**.

1738 **Service-directory-service** is a **service**

### 1740 5.23.3 Description

1741 Each concrete implementation of this specification will provide a **service-directory-service**. The **service-directory-**  
 1742 **service** will provide a simple registry for the **service** descriptions. Each realization of the **service-directory-service**  
 1743 will provide agents with a **service-root**, which will take the form of a set of **service-locators** including at least one  
 1744 **service-directory-service** (pointing to itself) In general, a **service-root** will provide sufficient entries to either describe  
 1745 all of the services available within the environment directly, or it will provide pointers to further services which will  
 1746 describe these services.

1747  
 1748 The following set of actions may be exposed by a **service-directory-service**. Each of these actions is optional.

### 1750 5.23.4 Actions

#### 1751 5.23.4.1 Register

1752 A service may **register** a **service** description in the form of a **service-directory-entry** with a **service-directory-**  
 1753 **service**.

1754  
 1755 The semantics of this action are as follows:

The **service** provides a **service-directory-entry** that is to be registered. In initiating the action, the **service** may control the scope of the action. Different reifications may handle this in different ways. The action may be addressed to a particular instance of a **service-directory-service**, or the action may be qualified with some scope parameter.

If the action is successful, the **service-directory-service** will return an **action-status** indicating success. Following a successful **register**, the **service-directory-service** will support legal **deletederegister**, and **querysearch** actions with respect to the registered **service-directory-entry**.

If the action is unsuccessful, the **service-directory-service** will return an **action-status** indicating failure, together with an **explanation**. The range of possible explanations is, in general, specific to a particular reification. However a conforming reification must, where appropriate, distinguish between the following explanations:

- Duplicate – the new entry “clashed” with some existing **service-directory-entry**.
- Access – the **agent** or **service** making the request is not authorized to perform the specified action.
- Invalid – the **service-directory-entry** is invalid in some way.

#### 5.23.4.2 **De**register/ete

A **service** may **deletederegister** a **service-directory-entry** from a **service-directory-service**. The semantics of this action are as follows:

The **service** provides a **service-directory-entry** which has the same **service-id/service-name** as that which is to be **deletedderegistered**. (The rest of the **service-directory-entry** is not significant.) In initiating the action, the **service** may control the scope of the action. Different reifications may handle this in different ways. The action may be addressed to a particular instance of a **service-directory-service**, or the action may be qualified with some scope parameter.

If the action is successful, the **service-directory-service** will return an **action-status** indicating success. Following a successful **deletederegister**, the **service-directory-service** will no longer support **modify**, **deletederegister**, and **querysearch** actions with respect to the **deleted\_deregistered service-directory-entry**.

If the action is unsuccessful, the **service-directory-service** will return an **action-status** indicating failure, together with an **explanation**. The range of possible explanations is, in general, specific to a particular reification. However a conforming reification must, where appropriate, distinguish between the following explanations:

- Not-found – the new entry did not match any existing **service-directory-entry**. This would only occur if no existing **service-directory-entry** had the same **service-id/service-name**
- Access – the **agent** or **service** making the request is not authorized to perform the specified action.
- Invalid – the **service-directory-entry** is invalid in some way.

#### 5.23.4.3 **QuerySearch**

A **service** or **agent** may **querysearch** a **service-directory-service** to locate **service-directory-entries** of interest. The semantics of this action are as follows:

The **querysearching** entity (**agent**) provides a **service-directory-entry** that is to be treated as a search pattern. In initiating the action, the **agent** may control the scope of the action. Different reifications may handle this in different ways. The action may be addressed to a particular instance of a **service-directory-service**, or the action may be qualified with some scope parameter.

The directory service verifies that the argument is a valid **service-directory-entry**. It then searches for registered **service-directory-entries** that satisfy the search criteria. A registered entry satisfies the search criteria if there is a

1811 match between each **key-value pair** in the submitted entry. The semantics of “matching” are likely to be reification-  
 1812 dependent; at a minimum, there should be support for matching on the *same* value and on *any* value.  
 1813

1814 If the action is successful, the **service-directory-service** will return an **action-status** indicating success, together with  
 1815 a set of **service-directory-entries** that satisfy the search pattern. The mechanism by which multiple entries are  
 1816 returned, and whether or not an **agent** may limit or terminate the delivery of results, is not defined in the abstract  
 1817 architecture and is therefore reification dependent.  
 1818

1819 If the action is unsuccessful, the **service-directory-service** will return an **action-status** indicating failure, together with  
 1820 an **explanation**. The range of possible explanations is, in general, specific to a particular reification. However a  
 1821 conforming reification must, where appropriate, distinguish between the following explanations:  
 1822

- 1823 • Not-found – the search pattern did not match any existing **service-directory-entry**.
- 1824
- 1825 • Access – the **agent** or **service** making the request is not authorized to perform the specified action.
- 1826
- 1827 • Invalid – the **service-directory-entry** is invalid in some way.
- 1828

#### 1829 5.23.4.4 Modify

1830 A **service** may **modify** a **service-directory-entry** that has been registered with a **service-directory-service**. The  
 1831 semantics of this action are as follows:  
 1832

1833 The **service** provides a **service-directory-entry** which contains the same **service-id****service-name** as the entry to be  
 1834 modified. In initiating the action, the **service** may control the scope of the action. Different reifications may handle this  
 1835 in different ways. The action may be addressed to a particular instance of a **service-directory-service**, or the action  
 1836 may be qualified with some scope parameter.  
 1837

1838 The **service-directory-service** verifies that the argument is a valid **service-directory-entry**. It then searches for a  
 1839 registered **service-directory-entry** with the same **service-id****service-name**. If it does not find one, the action fails and  
 1840 an **explanation** provided. Otherwise it modifies the existing **service-directory-entry** by examining each **key-value-**  
 1841 **pair** in new **service-directory-entry**. If the **value** is non-null, the **key-value-pair** is added to the new entry, replacing  
 1842 any existing **key-value-pair** with the same **key** identity. If the **value** is null, any existing **key-value-pair** with the same  
 1843 **key** identity is removed from the entry.  
 1844

1845 If the action is successful, the **service-directory-service** will return an **action-status** indicating success, together with  
 1846 a **service-directory-entry** corresponding to the new contents of the registered entry. Following a successful **modify**,  
 1847 the **service-directory-service** will support legal **modify**, **deletederegister**, and **querysearch** actions with respect to  
 1848 the modified **service-directory-entry**.  
 1849

1849 If the action is unsuccessful, the **service-directory-service** will return an **action-status** indicating failure, together with  
 1850 an **explanation**. The range of possible explanations is, in general, specific to a particular reification. However a  
 1851 conforming reification must, where appropriate, distinguish between the following explanations:  
 1852

- 1853 • Not-found – the new entry did not match any existing **service-directory-entry**. This would only occur if no existing  
 1854 **service-directory-entry** had the same **service-id****service-name**
- 1855
- 1856 • Access – the **agent** or **service** making the request is not authorized to perform the specified action.
- 1857
- 1858 • Invalid – the new **service-directory-entry** is invalid in some way.
- 1859

## 5.24 Service Id

### 5.24.1 Summary

The **service-id****service-name** provides uniqueness preservation within a given namespace. The **service-id****service-name** is used to test for equivalence of a **service**, and for modifying, deleting and searching for **service-directory-entries** within a **service-directory-service**. **Service-nameids** are unique, and are intended only to be used to test for uniqueness and identity, not to provide location or other extrinsic properties of the service. A concrete instantiation of **service-id****service-name** is a mandatory element of every concrete instantiation of the abstract architecture.

### 5.24.2 Relationships to other elements

**Service-nameid** is used to identify a **service** within a **service-directory service**

**Service-nameid** is a component of a **service-directory entry**.

### 5.24.3 Description

A **service-id****service-name** is an immutable identifier (e.g. a GUID, Globally Unique Identifier) that is associated with the **service** at creation time or initial registration. Name issuing should occur in a way that tends to ensure global uniqueness. This may be achieved, for example, through employing an algorithm that generates the name with a sufficient degree of stochastic complexity such as to induce a vanishingly small chance of a name collision.

## 5.25 Service Location Description

### 5.25.1 Summary

A **service-location-description** is a set of one or more **key-value tuples**, each containing a **signature-type**, **service-signature** and a **service-address**. In general, any **agent** or **service** wishing to use the **service** must 'already know' how to operate the service. In particular, the **service-address** should be a data value of type known both to the agent that it may use to invoke actions from the service. A concrete instantiation of **service-location-description** is a mandatory element of every concrete instantiation of the abstract architecture.

### 5.25.2 Relationships to Other Elements

**Service-locator** contains one or more **service-location-descriptions**

**Service-location-description** contains **signature-type**

**Service-location-description** contains **service-signature**

**Service-location-description** contains **service-address**

**Service-location-description** is used by an **agent** to access a **service**

### 5.25.3 Description

A **service-location-description** is the parallel structure to a **transport-description** (which is a component of the **agent-locator**), that describes how to access a **service**. Each **service-location-description** contains a **service-signature** that defines how to call the service, a **signature-type** that type classifies the **service-signature** and a **service-address** that identifies the addressable location of the **service**.

## 5.26 Service Locator

### 5.26.1 Summary

A **service-locator** consists of the set of **service-location-descriptions**, which can be used to access and make use of a **service**. In general, any **agent** or **service** wishing to use the **service** must 'already know' how to operate the service. In particular, the **service-address** should be a data value of type known both to the agent that it may use to invoke actions from the service. A concrete instantiation of **service-locator** is a mandatory element of every concrete instantiation of the abstract architecture.

## 1904 5.26.2 Relationships to Other Elements

1905 **Service-locator** is a member of **service-directory-entry**, which is registered with a **service-directory-service**

1906 **Service-locator** contains one or more **service-location-descriptions**

1907 **Service-locator** is used by an **agent** to access a **service**

## 1909 5.26.3 Description

1910 A **service-locator** is the parallel structure to an **agent-locator**, which describes how to access a **service**. Each  
1911 **service-locator** includes all of the **service-location-descriptions** that may be used to access the associated **service**.  
1912

## 1913 5.27 Service Root

### 1914 5.27.1 Summary

1915 A **service-root** is a set of **service-directory-entries** made available to an **agent** at start-up. This is the mechanism by  
1916 which an **agent** can bootstrap lifecycle support services, such as **message-transport-services** and **agent-directory-**  
1917 **services**, to provide it with a connection to the outside environment. A concrete instantiation of **service-root** is a  
1918 mandatory element of every concrete instantiation of the abstract architecture.  
1919

### 1920 5.27.2 Relationships to Other Elements

1921 **Service-root** is used by an **agent** to bootstrap **services**

1922 **Service-root** is a set of **service-directory-entries**

1923 **Service-root** should contain a **service-directory-entry** for at least one **message-transport-service**

1924 **Service-root** should contain a **service-directory-entry** for at least one **agent-directory-service**

1925 **Service-root** should contain a **service-directory-entry** for at least one **service-directory-service**

### 1927 5.27.3 Description

1928 An **agent** must be provided with a **service-root** at initialization in order for it to be able to communicate with other  
1929 **agents** and **services**. Typically the provider of the **service-root** will be a **service-directory-service** which will supply  
1930 a set of service descriptions in the form of **service-directory-entries** for available agent lifecycle support services,  
1931 such as **message-transport-services**, **agent-directory-services** and **service-directory-services**. In general, a  
1932 **service-root** will provide sufficient entries to either describe all of the services available within the environment  
1933 directly, or it will provide pointers to further services which will describe these services.  
1934

## 1935 5.28 Service Signature

### 1936 5.28.1 Summary

1937 A **service-signature** is a Fully Qualified Name within an administered namespace that describes the binding signature  
1938 for a service. A concrete instantiation of **service-signature** is a mandatory element of every concrete instantiation of  
1939 the abstract architecture.

### 1940 5.28.2 Relationships to Other Elements

1941 **Service-signature** is a component of a **service-locator**

1942 **Service-signature** is qualified in terms of a **signature-type**

### 1944 5.28.3 Description

1945 Examples of **service-signatures** are:

1946  
1947 org.fipa.standard.service.java-rmi-binding

1948 org.omg.agent.idl-binding

1949 See **signature-type** for a description of these **service-signature** bindings.

## 1952 5.29 Service Type

### 1953 5.29.1 Summary

1954 A **service-type** is a **key-value-tuple**, defining the *type* of a **service**. The set of possible values will be administered,  
1955 according to the process defined for **key-value-tuples** and by the appropriate namespace authority. A concrete  
1956 instantiation of **service-type** is a mandatory element of every concrete instantiation of the abstract architecture.

### 1958 5.29.2 Relationships to Other Elements

1959 **Service-type** is a component of a **service-directory-entry**

1960 **Service-type** qualifies the *type* of a **service**

### 1962 5.29.3 Description

1963 **Service-type** is used to classify the **service** in terms of some administered namespace. The *type* provides a  
1964 contextual reference to **service** functionality. For example, the **service-address** component of the **service-locator**  
1965 uses **service-type** as a context for communication bindings.

## 1967 5.30 Signature Type

### 1968 5.30.1 Summary

1969 A **signature-type** is a **key-value-tuple** describing the *type* of a **service-signature**. A **signature-type** allows the  
1970 interpretation of a **service-locator**, by associating it with a type of method signature binding. A concrete instantiation of  
1971 **signature-type** is an optional element of concrete instantiations of the abstract architecture.

### 1972 5.30.2 Relationships to Other Elements

1973 **Signature-type** is a component of a **service-locator**

1974 **Signature-type** qualifies the *type* of a **service-signature**

1975 **Signature-type** qualifies the *type* of a **service-address**

### 1977 5.30.3 Description

1978 The **signature-type** keys access to the opaque portion of a **service-locator**. Examples of signatures are:

1979 5.30.3.1.1 org.fipa.standard.service.java-rmi -binding

1980 For this **signature-type**, the **service-signature** is the Java IDL of the Java method to be invoked and the **service-**  
1981 **address** is the URL for the target of the remote method invocation.

1983 5.30.3.1.2 org.omg.agent.idl-binding

1984 For this **signature-type**, the **service-signature** is the OMG CORBA IDL of the method to be invoked and the **service-**  
1985 **address** is the IOR of the remote object which is the target of the method invocation.

1987 **5.31 Transport**1988 **5.31.1 Summary**

1989 A **transport** is a particular **message** delivery service, such as a message transfer system, a datagram service, a byte  
 1990 stream, or a shared scratchboard. Abstractly, a **transport** is a delivery system selected by virtue of the **transport-**  
 1991 **description** used to deliver **messages** to an **agent**. A concrete instantiation of **transport** is a mandatory element of  
 1992 every concrete instantiation of the abstract architecture.  
 1993

1994 **5.31.2 Relationships to Other Elements**

1995 **Transport-description** can be mapped onto a **transport**  
 1996 **Message-transport-service** may use one or more **transports** to effect message delivery  
 1997 A **transport** may support one or more **transport-encodings**  
 1998

1999 **5.31.3 Description**

2000 The mapping from **transport-description** to **transport** must be consistent across all realizations. FIPA will administer  
 2001 ontology of transport names. Concrete specifications should define a concrete encoding for this ontology.  
 2002

2003 **5.32 Transport Description**2004 **5.32.1 Summary**

2005 A **transport-description** is a **key-value tuple** containing a **transport-type**, a **transport-specific-address** and zero or  
 2006 more **transport-specific-properties**. A concrete instantiation of **transport-description** is a mandatory element of  
 2007 every concrete instantiation of the abstract architecture.  
 2008

2009 **5.32.2 Relationships to Other Elements**

2010 **Transport-description** has a **transport-type**  
 2011 **Transport-description** has a set of **transport-specific-properties**  
 2012 **Transport-description** has a **transport-specific-address**  
 2013 **Agent-directory-entries** include one or more **transport-descriptions**  
 2014 **Envelopes** contain one or more **transport-descriptions**  
 2015

2016 **5.32.3 Description**

2017 **Transport-descriptions** are included in the **agent-directory-service**, describing where a registered agent may be  
 2018 contacted. They can be included in the **envelope** for a **transport-message**, to describe how to deliver the message. In  
 2019 addition, if a **message-transport-service** is implemented, **transport-descriptions** are used as input to the **message-**  
 2020 **transport-service** to specify characteristics for additional delivery requirements for the delivery of **messages** to an  
 2021 **agent**.

2022 **5.33 Transport Message**2023 **5.33.1 Summary**

2024 A **transport-message** is the object conveyed from **agent** to **agent**. It contains the **envelope** containing **transport-**  
 2025 **descriptions** for the sender and receiver(s) together with a **payload** containing the encoded **message**. A concrete  
 2026 instantiation of **transport-message** is a mandatory element of every concrete instantiation of the abstract architecture.  
 2027

### 2028 5.33.2 Relationships to Other Elements

2029 **Transport-message** contains a **payload**  
 2030 **Transport-message** contains an **envelope**  
 2031

### 2032 5.33.3 Description

2033 A concrete implementation may limit the number of receiving **transport-descriptions** in the **envelope** of a **transport-**  
 2034 **message**. It may also establish particular relationships between the **agent-name** or **agent-names** for the receiver(s) in  
 2035 the **payload**. For example, it may ensure that there is a one-to-one correspondence between **agent-names**. The  
 2036 important thing to convey from **agent** to **agent** is the **payload**, together with sufficient **transport-message** context to  
 2037 send a reply. A gateway service or other transformation mechanism may unpack and reformat a **transport-message**  
 2038 as part of its processing.  
 2039

## 2040 5.34 Transport Specific Address

### 2041 5.34.1 Summary

2042 A **transport-specific-address** is an address specific to a particular **transport-type**. The format and description of the  
 2043 address will be specific to this type. The address is used by a **transport-service** in conjunction with a **transport-type**  
 2044 to construct transport connections. A concrete instantiation of **transport-specific-address** is an mandatory element of  
 2045 every concrete instantiation of the abstract architecture.  
 2046

### 2047 5.34.2 Relationships to Other Elements

2048 A **transport-specific-address** is a component of a **transport-description**.  
 2049 A **transport-specific-address** is associated with a specific **transport-type**.  
 2050

### 2051 5.34.3 Description

2052 The **transport-specific-address** provides a resolvable location descriptor, specific to a given **transport-type**, which  
 2053 can be used by a **transport-service** to send and/or receive **messages**.  
 2054

## 2055 5.35 Transport Specific Property

### 2056 5.35.1 Summary

2057 A **transport-specific-property** is property associated with a **transport-type**. These properties are used by the  
 2058 **transport-service** to help it in constructing transport connections, based on the properties specified. A concrete  
 2059 instantiation of **transport-specific-property** is an optional element of every concrete instantiation of the abstract  
 2060 architecture.  
 2061

### 2062 5.35.2 Relationships to Other Elements

2063 **Transport-description** includes zero or more **transport-specific-properties**  
 2064

### 2065 5.35.3 Description

2066 The **transport-specific-properties** are not required for a particular **transport**. They may vary between **transports**.  
 2067

2068 **5.36 Transport Type**2069 **5.36.1 Summary**

2070 A **transport-type** describes the type of transport associated with a **transport-specific-address**. A concrete  
2071 instantiation of **transport-type** is a mandatory element of every concrete instantiation of the abstract architecture.  
2072

2073 **5.36.2 Relationships to Other Elements**

2074 **Transport-description** includes a **transport-type**  
2075

2076 **5.36.3 Description**

2077 FIPA will administer an **ontology** of **transport-types**. FIPA managed types will be flagged with the prefix of "FIPA-".  
2078 Specific realizations can provide experimental types, which will be prefixed "X-"  
2079

2080

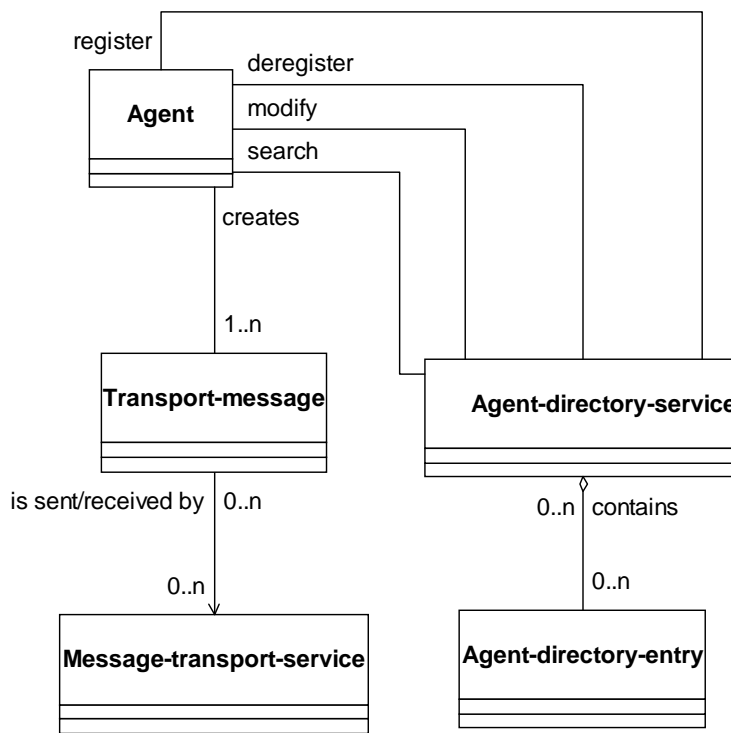
## 6 Agent and Agent Information Model

This section of the abstract architecture provides a series of UML class diagrams for key elements of the abstract architecture. In *Section 5, Architectural Elements* you can get a textual description of these elements and other aspects of the relationships between them.

**Comment on notation:** In UML, the notion of a 1 to many or 0 to many relationship is often noted as "1...\*" or "0...\*". However, the tool that was used to generate these diagrams used the convention "1...n" and "0...n" to express the concept of many.

### 6.1 Agent Relationships

*Figure 11* outlines the basic relationships between an **agent** and other key elements of the FIPA abstract architecture. In other diagrams in this section are provided details on the **agent-locator**, and the **transport-message**.

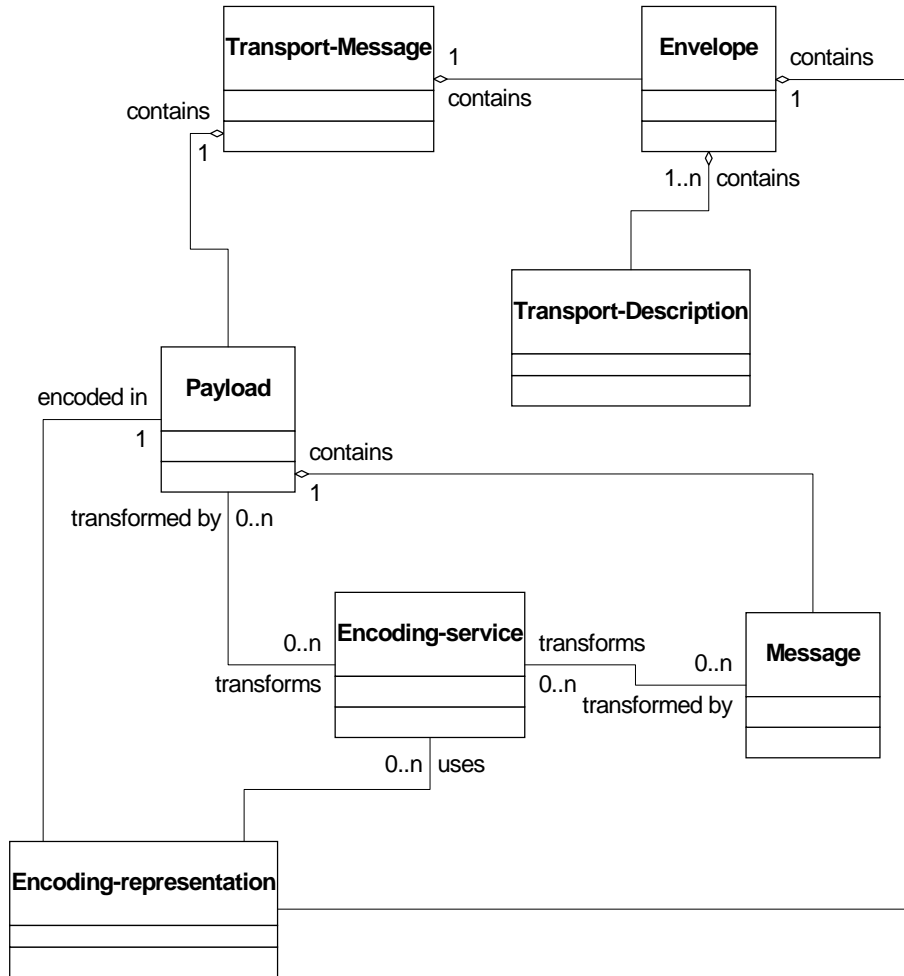


**Figure 11:** UML - Basic **Agent** Relationships

2096  
2097  
2098  
2099

## 6.2 Transport Message Relationships

**Transport-message** is the object conveyed from **agent** to **agent**. It contains the **transport-description** for the sender and receiver or receivers, together with a **payload** containing the **message** (see *Figure 12*).



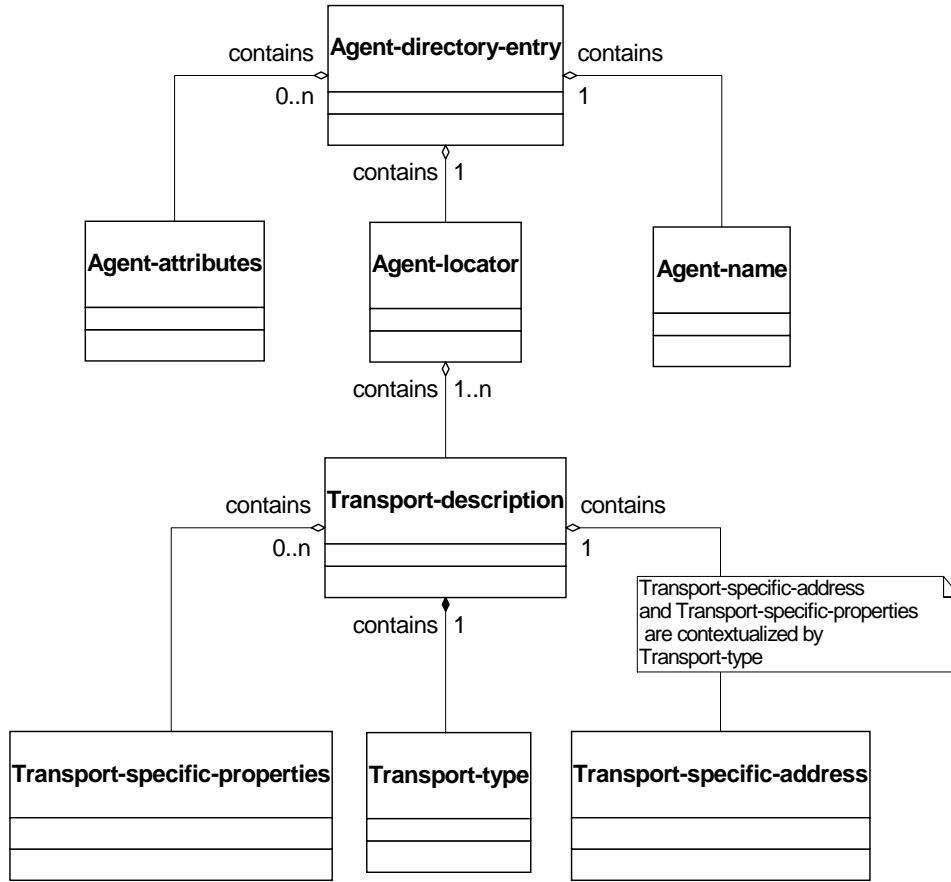
2100  
2101  
2102  
2103  
2104

Figure 12: UML - Transport-Message Relationships

2104  
2105  
2106  
2107

### 6.3 Agent Directory Entry Relationships

The **agent-directory-entry** contains the **agent-name**, **agent-locator** and **agent-attributes**. The **agent-locator** provides ways to address **messages** to an **agent**. It is also used in modifying **transport** requests (see *Figure 13*).



2108  
2109  
2110  
2111  
2112  
2113

Figure 13: UML - Agent-directory-entry and Agent-locator Relationships

### 6.4 Service Directory Entry Relationships

Figure 14 shows the hierarchical relationships within a **service-directory-entry** which contains the **service-id**, **service-name**, **service-type** and **service-locator**. The **service-locator** provides the means to contact and make use of a **service** and contains one or more **service-location-descriptions** which in turn each contain a **service-signature**, the **signature-type** and the **service-address**.

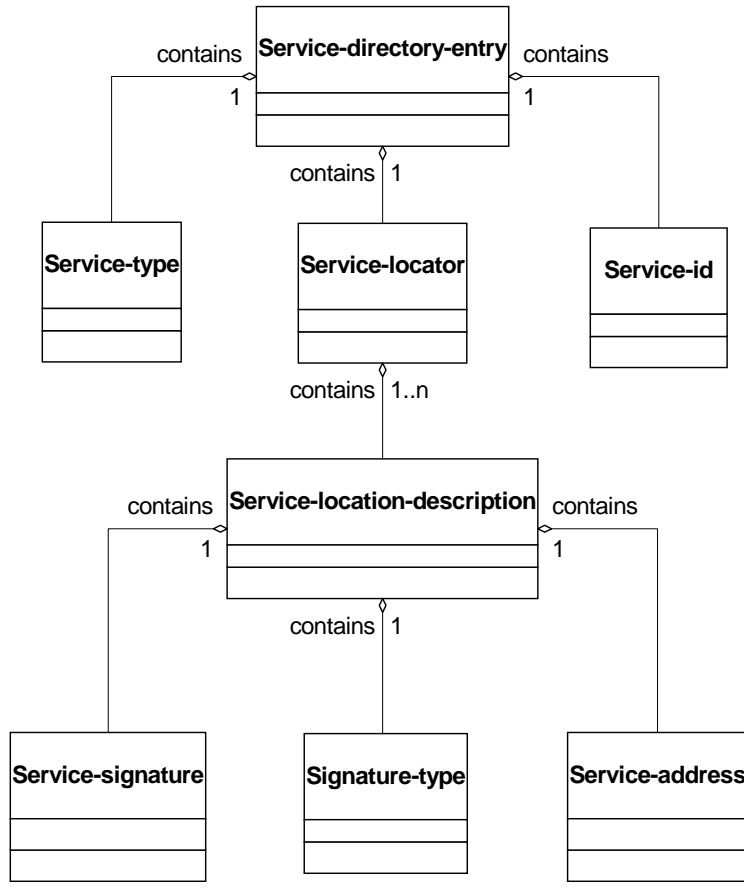


Figure 14: UML - Service-directory-entry and Service-locator Relationships

## 6.5 Message Elements

Figure 15 shows the elements in a **message**. A **message** is contained in a **transport-message** when messages are sent. Note that in Figure 14, the elements 'Communicative Act' and 'Performative' are not explicit architectural elements defined within this specification; they are informative entities relating to the semantics of the message as defined by the FIPA specification [FIPA00037]. Also, the multiplicity of the 'Ontologies' element refers to the fact more than one ontology may be referred to by the **ontology** architectural element which corresponds to the ACL message attribute 'Ontology' [FIPA00061].

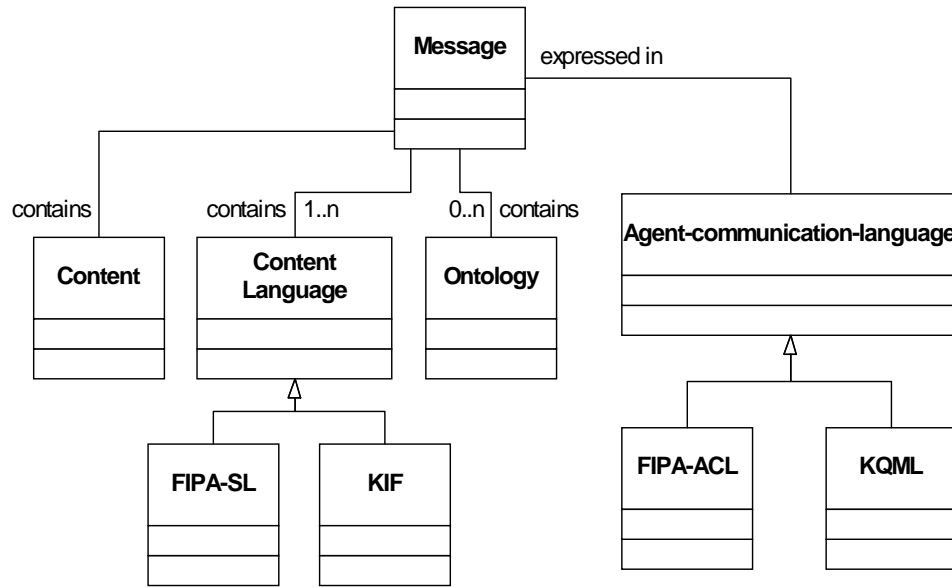
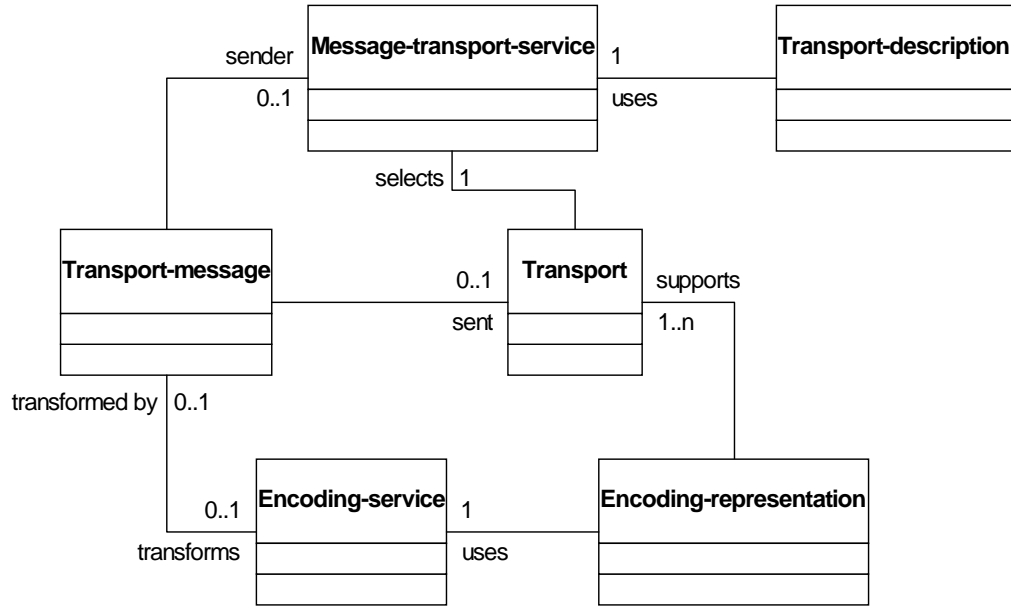


Figure 15: UML - Message Elements

2134  
2135  
2136  
2137

### 6.6 Message Transport Elements

The **message-transport-service** is an option service that can send **transport-messages** between **agents**. These elements may participate in other relationships as well (see *Figure 16*).



2138  
2139  
2140  
2141  
2142  
2143

Figure 16: UML - Message-Transport Elements

## 7 References

2143

2144 [FIPA00007] FIPA Content Language Library Specification. Foundation for Intelligent Physical Agents, 2000.  
2145 <http://www.fipa.org/specs/fipa00007/>

2146 [FIPA00023] FIPA Agent Management Specification. Foundation for Intelligent Physical Agents, 2000.

2147 <http://www.fipa.org/specs/fipa00023/>

2148 [FIPA00037] FIPA Communicative Act Library Specification. Foundation for Intelligent Physical Agents, 2000.

2149 <http://www.fipa.org/specs/fipa00037/>

2150 [FIPA00061] FIPA ACL Message Structure Specification. Foundation for Intelligent Physical Agents, 2000.

2151 <http://www.fipa.org/specs/fipa00061/>

2152 [Gamma95] Gamma, Helm, Johnson and Vlissides, Design Patterns. Addison-Wesley, 1995.

2153 [Searle69] Searle, J. L., Speech Acts. Cambridge University Press, 1969.

2154

2155

## 2155 8 Informative Annex A — Goals of Service Model

### 2156 8.1 Scope

2157 Agents require the use of many services in order to interoperate with other agents. In order to create the essential  
2158 abstractions for the various kinds of services that are essential to this mission, and to permit the straightforward  
2159 incorporation of other services in a consistent framework we require a model of services themselves.

### 2160 8.2 Variety of Services

2161 Although there are a number of essential services required by the abstract architecture, a fully built out platform may  
2162 include a wide variety of services not referenced in this document -- for example a platform may provide various kinds  
2163 of buffering services. Since the actual services may vary dynamically it is desirable for agents and services to have a  
2164 common model for discovering other services.

### 2165 8.3 Bootstrapping

2166 While the concrete realizations of the Abstract Architecture may have very different forms a common requirement  
2167 exists for many systems for a clear and reliable method of bootstrapping services, agents and agent systems.  
2168 Supporting bootstrapping is a clear aim of the service model  
2169

### 2170 8.4 Dynamic services

2171 The set of services available to an agent may on some systems be quite fixed: they are made available on start-up and  
2172 exist unchanged for the lifetime of the agent. However, on many – if not most – large scale systems, the set of services  
2173 available to agents is in fact dynamic. Both the number, type and instantiations of services are all is often subject to  
2174 change; for example, the message transport services available to an agent may vary depending on the circumstances.  
2175

2176 It is an objective of the service model to provide a consistent framework permitting services themselves to be made  
2177 dynamically available: services need to be able to dynamically register themselves, and agents and services may need  
2178 to be able to dynamically discover the appropriate services.  
2179

### 2180 8.5 Granularity

2181 An important – if informal – property of the service model is *granularity of services*. For example, it would may be  
2182 possible to `break apart` a message transport service into a collection of transports each of which is registered  
2183 independently with a service directory service. However, to do so would impose a significant burden on programmers  
2184 wishing to make use of message transport: a key benefit of supporting an integrated message transport service is that  
2185 it permits high-level convenience operations such as `reply to this message with this new message` or `send a  
2186 message to this agent` without requiring a `manual` search of the service directory service each time.  
2187

2188 In general the appropriate granularity of services depends on whether a range of related services is best viewed as  
2189 instantiations of a single high-level service or whether they are interdependent but distinct kinds of service.  
2190

### 2191 8.6 Example

2192 The following example illustrates how an entry in a service directory service can be formulated.  
2193

2194 For our example, we consider locating a prototype buffering service, implemented as Java object. The service, being  
2195 experimental, is contained within the name space, “org.fipa.experimental” and has the signature type “fipa-  
2196 experimental.buffer-prototype”.  
2197

2198 The Java object is accessed via the service address URL: `rmi://testbox.fipa.org/buffertest`  
2199

2200 The method signature is:  
2201 `public void setBuffer (BufferSessionContext ctx) throws java.rmi.RemoteException`

2202  
2203 So, we register the object by generating a service directory entry containing:

```
2204 (service-idservice-name, "org.BT.experimental.buffer-prototype.test-1")  
2205 (service-type, "org.fipa.experimental.buffer-prototype")  
2206 (service-locator, ((signature-type, " org.fipa.service-signature-ontology java2.rmi"),  
2207                   (service-signature, "fipa.agentpackages.experimentalbufferpackage"),  
2208                   (service-address, "rmi://testbox.Norwich.bt.co.uk/1066/buffertest")))
```

2210  
2211 The service-locator contains the signature-type which tells us that we use Java2 RMI to access the service. This tells  
2212 us how to understand the next two elements of the locator, the service-signature and service-address. The service-  
2213 signature is the Java package which you need to use to get at the methods provided by the buffering object. Finally,  
2214 the service-address is the resolvable location at which the appropriate method can be found.

2215  
2216  
2217  
2218

## 2218 9 Informative Annex B — Goals of Message Transport Service Abstraction

### 2219 9.1 Scope

2220 In order to create abstractions for the various architectural elements, it is necessary to examine the kinds of systems  
 2221 and infrastructures that are likely targets of real implementations of the abstract architecture. In this section, we  
 2222 examine some of the ways in which concrete messaging and messaging transports may differ. Authors of concrete  
 2223 architectural specifications must take these issues into account when considering what end-to-end assumptions they  
 2224 can safely make. The goals describe below give the reader an understanding of the objectives the authors of the  
 2225 abstract architecture had in mind when creating this architecture.  
 2226

### 2227 9.2 Variety of Transports

2228 There are a wide variety of transport services that may be used to convey a message from one agent to another. The  
 2229 abstract architecture is neutral with respect to this variety. For any instantiation of the architecture, one must specify  
 2230 the set of transports that are supported, how new transports are added, and how interoperability is to be achieved. It is  
 2231 permissible for a particular concrete architecture to require that implementations of that architecture must support  
 2232 particular transports.  
 2233

2234 Different transports use a variety of different address representations. Instantiations of the message transport  
 2235 architecture may support mechanisms for validating addresses, and for selecting appropriate transport services based  
 2236 upon the form of address used. It is extremely undesirable for an agent to be required to parse, decode, or otherwise  
 2237 rely upon the format of an address.  
 2238

2239 The following are examples of transport services that may be used to instantiate this abstract architecture:  
 2240

- 2241 • Enterprise message systems such as those from IBM and Tibco.
- 2242
- 2243 • A Java Messaging System (JMS) service provider, such as Fiorano.
- 2244
- 2245 • CORBA IIOP used as a simple byte stream.
- 2246
- 2247 • Remote method invocation, using Java RMI or a CORBA-based interface.
- 2248
- 2249 • SMTP email using MIME encoding.
- 2250
- 2251 • XML over HTTP.
- 2252
- 2253 • Wireless Access Protocol.
- 2254
- 2255 • Microsoft Named Pipes.
- 2256

### 2257 9.3 Support for Alternative Transports within a Single System

2258 Many application programming environments offer developers a variety of network protocols and higher-level  
 2259 constructs from which to implement inter-process communications, and it is becoming increasingly common for  
 2260 services to be made available over several different communications frameworks. It is expected that some  
 2261 instantiations of the FIPA architecture will allow the developer or deployer of agent systems to advertise the availability  
 2262 of their services over more than one message transport.  
 2263

2264 For this reason, the notion of transport address is here generalized to that of *destination*. A destination is an object  
 2265 containing one or more transport addresses. Each address is represented in a format that describes (explicitly or

2266 implicitly) the set of transports for which it is usable. (The precise mapping from address to transport is left to the  
 2267 concrete specification, although in practice the mapping is likely to be one-to-one.)  
 2268

2269 In its simplest form, a destination may be a single address that unambiguously defines the transport for which it can be  
 2270 used.  
 2271

#### 2272 **9.4 Desirability of Transport Agnosticism**

2273 The abstract architecture is consistent with concrete architectures which provide "transport agnostic" services. Such  
 2274 architectures will provide a programming model in which agents may be more or less aware of the details of transports,  
 2275 addressing, and many other communications-related mechanisms. For example, one agent may be able to address  
 2276 another in terms of some "social name", or in terms of service attributes advertised through the agent directory service  
 2277 without being aware of addressing format, transport mechanism, required level of privacy, audit logging, and so forth.  
 2278

2279 Transport agnosticism may apply to both senders and recipients of messages. A concrete architecture may provide  
 2280 mechanisms whereby an agent may delegate some or all of the tasks of assigning transport addresses, binding  
 2281 addresses to transport end-points, and registering addresses in white-pages or yellow-pages directories to the agent  
 2282 platform.  
 2283

#### 2284 **9.5 Desirability of Selective Specificity**

2285 While transport agnosticism simplifies the development of agents, there are times when explicit control of specific  
 2286 aspects of the message transport mechanism is required. A concrete architecture may provide programmatic access  
 2287 to various elements in the message transport subsystem.  
 2288

#### 2289 **9.6 Connection-Based, Connectionless and Store-and-Forward Transports**

2290 The abstract architecture is compatible with connection-based, connectionless, and store-and-forward transports. For  
 2291 connection-based transports, an instantiation may support the automatic reestablishment of broken connections. It is  
 2292 desirable that instantiations that implement several of these modes of operation should support transport-agnostic  
 2293 agents.  
 2294

#### 2295 **9.7 Conversation Policies and Interaction Protocols**

2296 The abstract architecture specifies a set of abstract objects that allows for the explicit representation of "a  
 2297 conversation", i.e. a related set of messages between interlocutors that are logically related by some interaction  
 2298 pattern. It is desirable that this property be achieved by the minimum of overhead at the infrastructure or message  
 2299 level; in particular, it is important that interoperability remain un-compromised. For example, an implementation may  
 2300 deliver messages to conversation-specific queues based on an interpretation of the message envelope. To achieve  
 2301 interoperability with an agent that does not support explicit conversations (i.e. which does not allow individual  
 2302 messages to be automatically associated with a particular higher-level interaction pattern), it is necessary to specify  
 2303 the way in which the message envelope must be processed in order to preserve conversational semantics.  
 2304

2305 *Note:* in the practice, we were not able to fully meet this goal. It remains a topic of future work.  
 2306

#### 2307 **9.8 Point-to-Point and Multiparty Interactions**

2308 The abstract architecture supports both point-to-point and multiparty message transport. For point-to-point interactions,  
 2309 an agent sends a message to an address that identifies a single receiving agent. (An instantiation may support implicit  
 2310 addressing, in which the destination is derived from the name of the intended recipient agent without the explicit  
 2311 involvement of the sender.) For multiparty message transport, the address must identify a group of recipients. The  
 2312 most common model for such message transport is termed "publish and subscribe", in which the address is a "topic" to  
 2313 which recipients may subscribe. Other models, for example, "address lists", are possible.

2314  
2315  
2316  
2317  
2318  
2319

Not all transport mechanisms support multiparty communications, and concrete architectures are not required to provide multiparty messaging services. Concrete architectures that do provide such services may support proxy mechanisms, so that agents and agent systems that only use point-to-point communications may be included in multiparty interactions.

## 2320 **9.9 Durable Messaging**

2321  
2322  
2323  
2324

Some commercial messaging systems support the notion of durable messages, which are stored by the messaging infrastructure and may be delivered at some later point in time. It is desirable that a message transport architecture should take advantage of such services.

## 2325 **9.10 Quality of Service**

2326  
2327  
2328

The term quality of service refers to a collection of service attributes that control the way in which message transport is provided. These attributes fall into a number of categories:

2329  
2330  
2331  
2332  
2333  
2334  
2335  
2336  
2337  
2338  
2339  
2340  
2341  
2342

- Performance,
- Security,
- Delivery semantics,
- Resource consumption,
- Data integrity,
- Logging and auditing, and,
- Alternate delivery.

2343  
2344  
2345  
2346  
2347  
2348

Some of these attributes apply to a single message; others may apply to conversations or to particular types of message transport. Architecturally it is important to be able to determine what elements of quality of service are supported, to express (or negotiate) the desired quality of service, to manage the service features which are controlled via the quality of service, to relate the specified quality of service to a service performance guarantee, and to relate quality of service to interoperability specifications.

## 2349 **9.11 Anonymity**

2350  
2351  
2352  
2353  
2354  
2355  
2356

The abstract transport architecture supports the notion of anonymous interaction. Multiparty message transport may support access by anonymous recipients. An agent may be able to associate a transient address with a conversation, such that the address is not publicly registered with any agent management system or directory service; this may extend to guarantees by the message transport service to withhold certain information about the principal associated with an address. If anonymous interaction is supported, an agent should be able to determine whether or not its interlocutor is anonymous.

## 2357 **9.12 Message Encoding**

2358  
2359  
2360  
2361

It is anticipated that FIPA will define multiple message encodings together with rules governing the translation of messages from one encoding to another. The message transport architecture allows for the development of instantiations that use one or more message encodings.

2362

### **9.13 Interoperability and Gateways**

2363

2364

2365

2366

2367

2368

2369

2370

The abstract agent transport architecture supports the development of instantiations that use transports, encodings, and infrastructure elements appropriate to the application domain. To ensure that heterogeneity does not preclude interoperability, the developers of a concrete architecture must consider the modes of interoperability that are feasible with other instantiations. Where direct end-to-end interoperability is impossible, impractical or undesirable, it is important that consideration be given to the specification of gateways that can provide full or limited interoperability. Such gateways may relay messages between incompatible transports, may translate messages from one encoding to another, and may provide quality-of-service features supported by one party but not another.

2371

### **9.14 Reasoning about Agent Communications**

2372

2373

2374

2375

The agent transport architecture supports the notion of agents communicating and reasoning about the message transport process itself. It does not, however, define the ontology or conversation patterns necessary to do this, nor are concrete architectures required to provide or accept information in a form convenient for such reasoning.

2376

### **9.15 Testing, Debugging and Management**

2377

2378

2379

2380

2381

In general, issues of testing, debugging, and management are implementation-specific and will not be addressed in an abstract architecture. Individual instantiations may include specific interfaces, actions, and ontologies that relate to these issues, and may specify that these features are optional or normative for implementations of the instantiation.

## 2381 **10 Informative Annex C — Goals of Directory Service Abstractions**

2382 This section describes the requirements and architectural elements of the abstract Directory Service. The directory  
 2383 service is that part of the FIPA architecture which allows agents to register information about themselves in one or  
 2384 more repositories, for those same agents to modify and ~~delete~~register this information, and for agents to search the  
 2385 repositories for information of interest to them. The information that is stored is referred to a directory entry, and the  
 2386 repository is an agent directory.  
 2387

### 2388 **10.1 Scope**

2389 The purpose of the abstract architecture is to identify the key abstractions that will form the basis of all concrete  
 2390 architectures. As such, it is necessarily both limited and non-specific. In this section, we examine some of the ways in  
 2391 which concrete directory services may differ.  
 2392

### 2393 **10.2 Variety of Directory Services**

2394 There are several directory services that may be used to store agent descriptions. The abstract architecture is neutral  
 2395 with respect to this variety. For any instantiation of the architecture, one must specify the set of directory services that  
 2396 are supported, how new directory services are added, and how interoperability is to be achieved. It is permissible for a  
 2397 particular concrete architecture to require that implementations of that architecture must support particular directory  
 2398 services.  
 2399

2400 Different directory services use a variety of different representations for schemas and contents. Instantiations of the  
 2401 agent directory architecture may support mechanisms for hiding these differences behind a common API and  
 2402 encoding, such as the Java JNDI model or hyper-directory schemes. It is extremely undesirable for an agent to be  
 2403 required to parse, decode, or otherwise rely upon different information encodings and schemas.  
 2404

2405 The following are examples of directory systems that may be used to instantiate the abstract directory service:

- 2406 • LDAP,
- 2407
- 2408 • NIS or NIS+,
- 2409
- 2410 • COS Naming,
- 2411
- 2412 • Novell NDS,
- 2413
- 2414 • Microsoft Active Directory,
- 2415
- 2416 • The Jini lookup service, and,
- 2417
- 2418 • A name service federation layer, such as JNDI.
- 2419
- 2420

### 2421 **10.3 Desirability of Directory Agnosticism**

2422 The abstract architecture is consistent with concrete architectures which provide "directory agnostic" services. Such a  
 2423 model will support agents that are more or less completely unaware of the details of directory services. A concrete  
 2424 architecture may provide mechanisms whereby an agent may delegate some or all of the tasks of assigning transport  
 2425 addresses, binding addresses to transport end-points, and registering addresses in all available directories to the  
 2426 agent platform.  
 2427

#### 2428 **10.4 Desirability of Selective Specificity**

2429 While directory agnosticism simplifies the development of agents, there are times when explicit control of specific  
2430 aspects of the directory mechanism is required. A concrete architecture may provide programmatic access to various  
2431 elements in the directory subsystem.  
2432

#### 2433 **10.5 Interoperability and Gateways**

2434 The abstract directory architecture supports the development of instantiations that use directory services appropriate to  
2435 the application domain. To ensure that heterogeneity does not preclude interoperability, the developers of a concrete  
2436 architecture must consider the modes of interoperability that are feasible with other instantiations. Where direct end-to-  
2437 end interoperability is impossible, impractical or undesirable, it is important that consideration be given to the  
2438 specification of gateways that can provide full or limited interoperability. Such gateways may extract agent descriptions  
2439 from one directory service, transform the information if necessary, and publish it through another directory service.  
2440

#### 2441 **10.6 Reasoning about Agent Directory**

2442 The abstract directory architecture supports the notion of agents communicating and reasoning about the directory  
2443 service itself. It does not, however, define the ontology or conversation patterns necessary to do this, nor are concrete  
2444 architectures required to provide or accept information in a form convenient for such reasoning.  
2445

#### 2446 **10.7 Testing, Debugging and Management**

2447 In general, issues of testing, debugging, and management are implementation-specific and will not be addressed in an  
2448 abstract architecture. Individual instantiations may include specific interfaces, actions, and ontologies that relate to  
2449 these issues, and may specify that these features are optional or normative for implementations of the instantiation.  
2450  
2451  
2452

## 11 Informative Annex D — Goals for Security and Identity Abstractions

### 11.1 Introduction

In order to create abstractions for the various architectural elements, it is necessary to examine the kinds of systems and infrastructures that are likely targets of real implementations of the abstract architecture. In this section, we examine some of the ways in which security related issues may differ. Authors of concrete architectural specifications must take these issues into account when considering what end-to-end assumptions they can safely make. The goals describe below give the reader an understanding of the objectives the authors of the abstract architecture had in mind when creating this architecture.

In practice, only a very minor part of the security issues can be addressed in the abstract architecture, as most security issues are tightly coupled to their implementation.

In general, the amount of security required is highly dependent on the target deployment environment.

A glossary of security terms is located at the end of this section.

### 11.2 Overview

There are several aspects to security, which must permeate the FIPA architecture. They are:

- **Identity.** The ability to determine the identity of the various entities in the system. By identifying an entity, another entity interacting with it can determine what policies are relevant to interactions with that entity. Identity is based on credentials, which are verified by a Credential Authority.
- **Access Permissions.** Based on the identity of an entity, determine what policies apply to the entity. These policies might govern resource consumption, types of file access allowed, types of queries that can be performed, or other controlling policies.
- **Content Validity.** The ability to determine whether a piece of software, a message, or other data has been modified since being dispatched by its originating source. Digitally signing data and then having the recipient verify the contents are unchanged often accomplish this. Other mechanisms such as hash algorithms can also be applied.
- **Content Privacy.** The ability to ensure that only designated identities can examine software, a message or other data. To all others the information is obscured. This is often accomplished by encrypting the data, but can also be accomplished by transporting the data over channels that are encrypted.

Identity, or the use of credentials, is needed to supply the ability to control access, to provide content validity, and create content privacy. Each of these is discussed below.

### 11.3 Areas to Apply Security

This section describes the areas in which security can be applied within agent systems. In each case, the security related risks that are being guarded against are described. The assumption is that any agent or other entity in the system may have credentials that can be used to perform various forms of validation.

#### 11.3.1 Content Validity and Privacy During Message Transport

There are two basic potential security risks when sending a message from one agent to another.

2499 The primary risk is that a message is intercepted, and modified in some way. For example, the interceptor software  
 2500 inserts several extra numbers into a payment amount, and modifies the name of the check payee. After modification, it  
 2501 is sent on to the original recipient. The other agent acts on the incorrect data. In a case like this, the *content* validity of  
 2502 the message is broken.

2503  
 2504 The secondary risk is that the message is read by another entity, and the data in it is used by that entity. The message  
 2505 does reach its original destination intact. If this occurs, the privacy of the message is violated.

2506  
 2507 Digital signing and encryption can address these risks, respectively. These two techniques can be abstractly presented  
 2508 at two different layers of the architecture. The messages themselves (or probably just the **payload** part) can be signed  
 2509 or encrypted. There are a number of techniques for this, PGP signing and encryption, Public Key signing and  
 2510 encryption, one time transmission keys, and other cryptographic techniques. This approach is most effective when the  
 2511 nature of underlying message transport is unknown or unreliable from a security perspective.

2512  
 2513 The message transport itself can also provide the digital signing or encryption. There are a number of transports that  
 2514 can provide such features: SKIP, IPSEC and CORBA Common Secure Interoperability Services. It seems prudent to  
 2515 include both models within the architecture, since different applications and software environments will have very  
 2516 different capabilities.

2517  
 2518 There is another aspect of message transport privacy that comes from agents that misrepresent themselves. In this  
 2519 scenario, an agent can register with directory services indicating that is a provider of some service, but in fact uses the  
 2520 data it receives for some other purpose. To put it differently, how do you know *who* you are talking to? This topic is  
 2521 covered under agent identity below.

### 2523 11.3.2 Agent Identity

2524 If agents and agent services have a digital identity, then agents can validate that:

- 2525 • Agents they are exchanging messages with can be accurately identified, and,
- 2526 • Services they are using are from a known, safe source.

2527  
 2528 Similarly, services can determine whether the agent:

- 2529 • Use identity to determine code access or access control decisions, or,
- 2530 • Use agent identity for non-repudiation of transactions.

### 2536 11.3.3 Agent Principal Validation

2537 The Agent can contain a principal (for example a user), on whose behalf this code is running. The principal has one or  
 2538 more credentials, and the credentials may have one or more roles that represent the principal.

2539  
 2540 If an agent has a principal, the other agents can:

- 2541 • Determine whether they want to interoperate with that agent,
- 2542 • Determine what policy and access control to permit to that user, and,
- 2543 • Use the identity to perform transactions.

2544  
 2545 Services could perform similar actions.

#### 2550 **11.3.4 Code Signing Validation**

2551 An agent can be code signed. This involves digitally signing the code with one or more credentials. If an agent is code  
2552 signed, the platform could:

- 2553 • Validate the credential(s) used to sign the agent software. Credentials are validated with a credential authority,
- 2554 • If the credentials are valid, use policy to determine what access this code will have, or,
- 2555 • If the credentials are valid, verify that the code is not modified.

2556 In addition, the Agent Platform can use the lack of digital signature to determine whether to allow the code to run, and  
2557 policy to determine what access the code will have. In other words, some platforms may have the policy that will not  
2558 permit code to run, or will restrict Access Permissions unless it is digitally signed.

### 2564 **11.4 Risks Not Addressed**

2565 There are a number of other possible security risks that are not addressed, because they are general software issues,  
2566 rather than unique or special to agents. However, designers of agent systems should keep these issues in mind when  
2567 designing their agent systems.

#### 2569 **11.4.1 Code or Data Peeping**

2570 An entity can probe the running agent and extract useful information.

#### 2572 **11.4.2 Code or Data Alteration**

2573 The unauthorized modification or corruption of an agent, its state, or data. This is somewhat addressed by the code  
2574 signing, which does not cover all cases.

#### 2576 **11.4.3 Concerted Attacks**

2577 When a group of agents conspire to reach a set of goals that are not desired by other entities. These are particularly  
2578 hard to guard against, because several agents may co-operate to create a denial of service attack in a feint to allow  
2579 another agent to undertake the undesirable action.

#### 2581 **11.4.4 Copy and Replay**

2582 An attempt to copy an agent or a message and clone or retransmit it. For example, a malicious platform creates an  
2583 illegal copy, or a clone, of an agent, or a message from an agent is illegally copied and retransmitted.

#### 2585 **11.4.5 Denial of Service**

2586 In a denial-of-service the attackers try to deny resources to the platform or an agent. For example, an agent floods  
2587 another agent with requests and the receiving agent is unable to provide its services to other agents.

#### 2589 **11.4.6 Misinformation Campaigns**

2590 The agent, platform, or service misrepresents information. This includes lying during negotiation, deliberately  
2591 representing another agent, service or platform as being untrustworthy, costly, or undesirable.

2593 **11.4.7 Repudiation**

2594 An agent or agent platform denies that it has received/sent a message or taken a specific action. For example, a  
 2595 commitment between two agents as the result of a contract negotiation is later ignored by one of the agents, denying  
 2596 the negotiation has ever taken place and refusing to honour its part of the commitment.  
 2597

2598 **11.4.8 Spoofing and Masquerading**

2599 An unauthorized agent or service claims the identity of another agent or piece of software. For example, an agent  
 2600 registers as a Directory Service and therefore receives information from other registering agents.  
 2601

2602 **11.5 Glossary of Security Terms**

2603 **Access permission** – Based on a credential model, the ability to allow or disallow software from taking an action. For  
 2604 example, software with certain credentials may be allowed read a particular file, a group with different credentials may  
 2605 be allowed to write to the file.

2606 *Examples: OS file system permissions, Java Security Profiles (check name), Database access controls.*  
 2607

2608 **Authentication** – Using some credential model, ability to verify that the entity offering the credentials is who/what it  
 2609 says it is.  
 2610

2611 **Credential** – An item offered to prove that a user, a group, a software entity, a company, or other entities is who or  
 2612 what it claims to be.

2613 *Examples: X.509 certificate, a user login and password pair, a PGP key, a response/challenge key, a fingerprint, a*  
 2614 *retinal scan, a photo id. (Obviously, some of these are better suited to software than others!)*  
 2615

2616 **Credential Authority** – An entity that determines whether the credential offered is valid, and that the credential  
 2617 accurately identifies the individual offering it.

2618 *Examples: An X.509 certificate can be validated by a certificate authority. At a bar, the bartender is the credential*  
 2619 *authority who determines whether your photo id represents you (he may then determine your access permissions to*  
 2620 *available beverages!).*  
 2621

2622 **Credential model** – The particular mechanism(s) being used to provide and authenticate credentials.  
 2623

2624 **Code signing** – A particular case of digital signature (see below), where code is signed by the credentials of some  
 2625 entity. The purpose of code signing is to identify the source of the code, and to verify that the code has not been  
 2626 changed by another entity.

2627 *Examples: Java code signing, DCOM object signing, checksum verification.*  
 2628

2629 **Digital signature** – Using a credential model to indicate the source of some data, and to ensure that the data is  
 2630 unchanged since it was signed. Note: the word data is used very broadly here – it could a string, software, voice  
 2631 stream, etc.

2632 *Examples: S/MIME mail, PGP digital signing, IPSEC (authentication modes)*  
 2633

2634 **Encryption** – The ability to transform data into a format that can only be restored by the holder of a particular  
 2635 credential. Used to prevent data from being observed by others.

2636 *Examples: SSL, S/MIME mail, PGP digital signing, IPSEC (encryption modes)*  
 2637

2638 **Identity** – A person, server, group, company, software program that can be uniquely identified. Identities can have  
 2639 credentials that identify them.  
 2640

2641 **Lease** – An interval of time that some element, such as an identity or a credential is good for. Leases are very useful  
 2642 when you want to restrict the length of commitment. For example, you may issue a temporary credential to an agent  
 2643 that gives it 20 minutes in a given system, at which time the credential expires.  
 2644

2645 **Policy** – Some set of actions that should be performed when a set of conditions is met. In the context of security, allow  
2646 access permissions based on a valid credential that establishes an identity.

2647 *Examples: If a credential for a particular user is presented, allow him to access a file. If a credential for a particular role*  
2648 *is presented, allow the agent to run with a low priority.*

2649

2650 **Role** – An identity that has an "group" quality. That is, the role does not uniquely identify an individual, or machine, or  
2651 an agent, but instead identifies the identity in a particular context: as a system manager, as a member of the entry  
2652 order group, as a high-performance calculation server, etc.

2653 *Examples: In various operating system groups, as applied to file system access. In Lotus Notes, the "role" concept.*  
2654 *X.509 certificate role attributes.*

2655

2656 **Principal** – In the agent domain, the identity on whose behalf the agent is running. This may be a user, a group, a role  
2657 or another software entity.

2658 *Examples: A shopping agent's principal is the user who launched it. An commodity trader agent's principal is a*  
2659 *financial company. A network management agent's principal is the role of system admin, or super-user. In a small*  
2660 *"worker bee" agent, the principal may be the delegated authority of the parent agent.*

2661

2662

## 2662 12 Informative Annex E — ChangeLog

### 2663 12.1 2001/11/01 - version I by TC Architecture

2664	All document	<b>directory-service</b> becomes <b>agent-directory-service</b> .
2665	All document	<b>directory-entry</b> becomes <b>agent-directory-entry</b> .
2666	All document	<b>locator</b> becomes <b>agent-locator</b> .
2667	All document	<b>Encoding-transform-service</b> becomes <b>encoding-service</b> .
2668		
2669	Section 1, Paragraph 5	Note added concerning availability of documents.
2670	Section 1.1	Annexes updated to include new ones.
2671	Section 2.1	Changed text of second bullet point.
2672	Section 2.1	Section descriptions updated to include new annexes.
2673	Section 2.3, Paragraph 2	Added complete paragraph.
2674	Section 4.1, Paragraph 1	Changed 2nd sentence changed to include <b>service-directory-service</b> .
2675	Section 4.1, Paragraph 2	First sentence added.
2676	Section 4.2	Added complete section.
2677	Section 4.3	Table updated to correct <b>agent-locator</b> description.
2678	Section 4.3.1	Changed section heading.
2679	Section 4.3.2	Changed section heading.
2680	Section 4.4	Added complete section.
2681	Section 4.5, Paragraph 1	Changed "fundamental aspects" to include message representation.
2682	Section 4.5.1, Paragraph 1	Replaced 3rd sentence.
2683	Section 4.5.1, Figure 6	Receiver (and <b>agent-name</b> for receiver) made plural.
2684	Section 4.5.2	Added complete section.
2685	Section 4.5.3, Figure 7	Receiver (and <b>agent-name</b> for receiver) made plural.
2686	Section 5.1.5, Table 2	Included Fully Qualified Name column for each element
2687		Changed description of <b>encoding-service</b> .
2688		Changed <b>service</b> presence to be mandatory.
2689		Added <b>service-address</b> .
2690		Added <b>service-attributes</b> .
2691		Added <b>service-directory-service</b> .
2692		Added <b>service-directory-entry</b> .
2693		Added <b>service-id</b> .
2694		Added <b>service-location-description</b> .
2695		Added <b>service-locator</b> .
2696		Added <b>service-root</b> .
2697		Added <b>service-signature</b> .
2698		Added <b>service-type</b> .
2699		Added <b>signature-type</b> .
2700		Added <b>transport-specific-address</b> .
2701	Section 5.2	Added complete section.
2702	Section 5.3	Added complete section.
2703	Section 5.4.2	Removed first point.
2704	Section 5.6.1, Paragraph 1	Removed 2nd and 3rd sentence. Added new 2nd sentence.
2705	Section 5.6.1, Paragraph 2	Removed.
2706	Section 5.6.2	Added new relationship.
2707	Section 5.10.3	Changed 1st sentence so that GUID now an example.
2708	Section 5.11.1	Changed 1st sentence to include <b>message</b> reference.
2709		Moved 2nd and 3rd sentences to Section 5.11.3
2710		Added new 2nd sentence.
2711	Section 5.11.2	Changed 2nd relationship to be more accurate.
2712	Section 5.11.3	Added complete section.
2713	Section 5.13.1, Paragraph 1	Changed 2nd sentence to include Bit-efficient encoding.

2714		Added 3rd sentence.
2715	Section 5.13.1, Paragraph 2	Removed.
2716	Section 5.13.2	Changed 1st relationship.
2717		Removed 2nd, 3rd and 4th relationships.
2718		Added new 2nd relationship.
2719	Section 5.14.1	Added 3rd sentence.
2720	Section 5.14.2	Changed 2nd, 3rd and 4th relationship.
2721		Removed 5th relationship.
2722	Section 5.14.3.1	Changed section heading.
2723	Section 5.14.3.1. Paragraph 1	Changed 1st and 2nd sentences.
2724	Section 5.14.3.1. Paragraph 2	Changed 1st sentence.
2725	Section 5.14.3.1. Paragraph 3	Added complete paragraph.
2726	Section 5.14.3.1	Added 'invalid payload' explanation.
2727	Section 5.14.3	Added new 2nd sentence.
2728	Section 5.14.3	Deleted last 2 sentences.
2729	Section 5.16.1	Added last sentence.
2730	Section 5.16.3	Changed 1st to include <b>service-directory-service</b> .
2731	Section 5.17.1	Added new 4th and last sentences.
2732	Section 5.17.1	Added 'and ontologies' to 6th sentence.
2733	Section 5.17.3	Updated final two relationships.
2734	Section 5.19.2	Updated both relationships with respect to <b>ontologies</b> .
2735	Section 5.21.2	Added three new relationships related to service model.
2736	Section 5.22	Added complete section.
2737	Section 5.23	Added complete section.
2738	Section 5.24	Added complete section.
2739	Section 5.25	Added complete section.
2740	Section 5.26	Added complete section.
2741	Section 5.27	Added complete section.
2742	Section 5.28	Added complete section.
2743	Section 5.29	Added complete section.
2744	Section 5.30	Added complete section.
2745	Section 5.31	Added complete section.
2746	Section 5.32	Added complete section.
2747	Section 5.36	Added complete section.
2748	Section 6.2, Figure 12	Changed <b>message-encoding-representation</b> to <b>encoding-representation</b> .
2749		Changed <b>transform-service</b> to <b>encoding-service</b> .
2750		Changed role linking <b>payload</b> and <b>message</b> .
2751		Removed role linking <b>transport-message</b> and <b>encoding-representation</b> .
2752		Removed role linking <b>transport-message</b> and <b>encoding-service</b> .
2753		Removed <b>payload-external-attributes</b> .
2754		Added role linking <b>envelope</b> and <b>encoding-representation</b> .
2755	Section 6.3, Figure 13	Changed role linking <b>agent-directory-service</b> and <b>agent-locator</b> from 'contains 1..n' to 'contain 1'.
2756		Changed role linking <b>agent-locator</b> and <b>transport-description</b> from 'contains 1' to 'contain 1..n'.
2757		Changed role linking transport-description and transport-type from "has a" to "contains 1".
2758		
2759		Changed role linking transport-description and transport-type from "has a" to "contains 1".
2760		
2761	Section 6.4	Added complete section.
2762	Section 6.5, Paragraph 1	Added final two sentences.
2763	Section 6.5, Figure 15	Changed role linking <b>message</b> and "communicative act" from 'contains 1..n' to 'is a'.
2764		Changed role linking "communicative act" and <b>content</b> from 'contains 1..n' to 'contains 1'.
2765		
2766	Section 7	Added reference for FIPA00095.
2767	Section 8	Added complete section.
2768	Section 9	Added complete section.
2769	Section 10	Added word 'service' into section heading.

2770 Section 13 Added complete section.  
2771

2772 **12.2 2002/05/02 - version K by FIPA Architecture Board**

2773 All document All instances of **service-id** rPage x, line y: ~~<blah>~~eplaced with **service-name**  
2774 for coherence with **agent-name**.  
2775 All document **Delete** action changed to **Deregister** for both **agent-directory-service** and **service-**  
2776 **directory-service**.  
2777 All document **Query** action changed to **Search** for both **agent-directory-service** and **service-**  
2778 **directory-service**.  
2779 Section 5.23.3 Note that all actions of the **service-directory-service** are optional.  
2780