Architectural Concepts for Agent Paradigm : A Way to Separate Concerns in Open Distributed Systems

A. Diagne
Université Pierre & Marie Curie
Laboratoire d'Informatique de Paris 6
Thème Systèmes Répartis et Coopératifs
4, place Jussieu 75252 Paris Cedex 05, France
Phone : (+33) (0)1 44 27 73 65, Fax : (+33) (0)1 44 27 62 86
E-mail : Alioune.Diagne@masi.ibp.fr

Abstract

The emerging agent paradigm is gaining legitimacy as a solution to the most and most complex needs in distributed systems. For instance, a new concept like remote programming is presented as an alternative to the limits of the classical client-server interaction modes including its derivates. Agent paradigm has also emphasized the concept of service as a set of functionalities with contractual constraints. Nevertheless, agent paradigm does not always deal with architectural concepts, it is mostly concerned with implementation of such advanced features in distributed systems. Meanwhile RM-ODP undertakes federation of open distributed systems with a generic architecture. This architecture is based on progressive structuration of systems within five viewpoints based upon object-oriented concepts. Viewpoints encompass representations from conceptual level to final implementation in a progressive way and supply a quite satisfactory basis for separation of the concerns. In this paper, we address the relevancy of such an architecture to the agent paradigm in order to separate the many concerns in open distributed systems (e.g. collaboration, cognition and reactivity).

Keywords

Objects, Agents, Service, Architecture for Open Distributed Systems, Separation of Concerns.

1 INTRODUCTION

Agent paradigm is gaining legitimacy as a solution to most and most complex needs in distributed systems (Magedanz 96). It brings concepts beyond the ones introduced by the object-oriented paradigm. These new concepts are presented as alternatives to limits of classical approaches in distributed systems like client-server and its various interworking modes (variations of the Remote Procedure Call, Message Passing, etc.).

The agent paradigm is nowadays supported efficiently by technological proposals (Sun 95), (Telescript 95). Most of the newly introduced concepts are supported efficiently at run-time and ongoing research is very active. Separation of Concerns is a new trend in software engineering which tries to formally separate and organize the many aspects one can have to handle through systems life-cycle (Hursch 95).

The RM-ODP is an international standard issued by ITU and ISO to overcome the necessity to put together heterogeneous systems in order to cooperate. Many vendors, each having a system (a set of services or a computing platform) with its own characteristics, must be able to cooperate by proposing or requesting some services. This can be made largely easier when systems share a common architectural basis. The RM-ODP is a proposal of a generic architecture for open distributed systems based upon concepts from the object-oriented paradigm. It proposes a prescriptive model and a descriptive one to structure such systems. As a generic standard, it is somewhat unprescriptive on some aspects which may vary with the application domain. However, it has achieved a first level of a federative approach for heterogeneous distributed systems.

Structuration and architecture are important research fields in agent-based systems. Many authors have already tackled the problem with separation of concerns as a major need to fullfil. Communication versus Knowledge Sharing is the basis of structuration in (Finin 94). The proposal in (Carle 95) focuses on the distributed artificial intelligence aspects like distributed problem solving and reduce reactive and collaborative aspects to a minimum. The proposal in (McKay 96) also stressed at the communication aspects for load sharing and knowledge sharing. Collaborative and reactive aspects are not considered. In (Pitt 95), the authors have proposed a satisfactory structuration - the Cooperative Services Framework - but the separation of concerns is not fully achieved. They fully consider the collaborative aspects with deontic logic but the cognitive aspects are forgotten and the reactive ones are not elaborated. In (Merz 96), the authors focus on the open infrastructure necessary to support openess of clients and servers cooperations. Petri net-based control flows which characterize a service allow to handle goal splitting and sub-agents creation with validation. A main remark from these works is that authors do not aim to apply formal engineering methods in their structuration. This aim needs a demarcation between the aspects that can be (and need to be) formally verified and those that can not. For these last ones, a validation by simulation can be helpful and sufficient.

The aim of this paper is to apply an ODP-like structuration to agent-based services within open distributed systems in order to separate the concerns in a way which enables formal engineering methods. The Section 2 is dedicated to a brief overview of ODP concepts while Section 3 presents some key ideas of the agent paradigm. Section 4 presents the new trend of service which nowadays influences the way to think about open distributed systems. In Section 5 we investigate the relevancy of an adaptation of ODP viewpoints to agent paradigm. In Section 6 we propose and discuss an architectural framework for ODP-like agent-based service in open distributed systems before conclusion in Section 7.

2 OVERVIEW OF THE RM-ODP CONCEPTS

The RM-ODP defines first a consistent set of modeling concepts on which is based afterwards the definition of the proposed architecture. It therefore organizes the concepts into two main parts :

- a *general framework* which is the *descriptive model* of a distributed system (ITU X.902),
- an *organization* and *structure* of a distributed system which is the *prescriptive model* (ITU X.903).

2.1 The Descriptive Model

The descriptive model itself contains:

- *basic modeling concepts* dealing with an object-based model like encapsulation, abstraction, interface, behavior, state, etc.
- *specification concepts* not intrinsic to distributed systems but including object-oriented notions of type and class, template, hierarchy, role, etc.
- *architectural concepts* which correspond to ubiquitous notions and structures in distributed systems conceptual and operational descriptions.

2.2 The Prescriptive Model

The prescriptive model is organized into five progressive viewpoints. Each viewpoint aims to capture part of the information processed in system specification and design.

- Enterprise Viewpoint : This viewpoint represents the information relative to the overall *rules and policy* of the target system. For complex systems, this first abstraction of a system may involve some contextual and contractual aspects with knowledge representation and inference. Components of the systems have *roles* to play, *obligations* to fullfil toward the other ones and *negotiations* to carry on in order to cooperate safely and reliably.
- Information Viewpoint : Here, one must consider further the semantics of the information stored and manipulated in the system. There are three schemas:
  - the *static schema* defines the state and the structure of system components,
  - the *invariant schema* establishes properties that must always be true whatever the components might evolve,
  - the *dynamic schema* specifies the behavior components can have with respect to the two previous ones.

- Computational Viewpoint : At this level, the main concern is the functional decomposition of the system into structures that fit for *distribution*. A system is considered as a set of *components*, each of them has *functionalities* it is able to offer to the others. These functionalities can be attached with some constraints like usage requirements and quality of service. The *interactions* between the many components are described as well as the mechanisms that allow to support them.
- Engineering and Technological Viewpoints : These two last viewpoints consider the *infrastructure* and the *technological choices* to support the three previous ones. They consider all what is necessary to support the processing, storage and communication activities depicted out so far. They also give means to optimize those activities for purpose of performances in the system.

2.3 Summary

The reference model proposes five progressive viewpoints. The three first ones deal with the structuration of a system and can be made independent from the underlining technology. The two last ones cope more with realization aspects. Consistency between the many viewpoints of a system must be managed (Bowman 96). However, in the current trend of open distributed systems, we can not make the viewpoints as separated as in the ODP approach. Providing services in the «*open electronic market*» encompass problems which make the mixing of the many viewpoints and their representation at the processing level necessary.

3 OVERVIEW OF THE AGENT PARADIGM

Agent paradigm emerges from artificial intelligence works (Ferber 95). It aims to enhance software entities (data attached with behavior according to object orientation) with knowledge storage and inference capabilities, mobility and adaptability, advanced cooperation and collaboration mechanisms among other aspects. Agent paradigm covers a large set of application domains and can be considered from different points of view (reactivity, cognition, etc.).

3.1 Characteristics of Agents

What is an agent ? It is somewhat difficult to find in the literature a definition which makes a consensus. According to Ferber, «*an agent is an hardware of software entity able to act on itself and on its environment. It has a partial representation of its environment and is able to communicate with other agents. It aims an individual goal and its behavior is the result of its observations, knowledge, abilities and interactions it can have with other agents and with the environment*» (Ferber 95). Agents are a self-contained entities which have several concerns to cope with. For this reason, agents are often tagged with the non-exhaustive following list of attributes :

- *intelligent* : an intelligent agent is able to have cognitive activities. An intelligent agent is a piece of software which behaves according to some attached knowledge and inference it can perform on it. There is a wide range of intelligence levels ranging from simple pre-defined rules to achieve indeterministic tasks to self-learning inference mechanisms. The knowledge of agents is often incomplete and they need to cooperate/collaborate with each other to enhance or refine it,

- *collaborative/cooperative* : agents are able to collaborate/cooperate with each other. Collaboration and cooperation are beyond communication and interaction because they encompass making agreement on the high level purpose of interactions, establishing the communication means and late supplying the context and making the binding necessary to the real execution. Cooperation is based on coordination of elementary activities in order to fullfil more elaborated ones. For instance, many servers can cooperate to enhance the quality of services they propose (e.g. fault tolerance by replication, best performances by competition, etc.). Collaboration is more a client/server notion based on contracts and proposal/request of services,
- *mobile* : a mobile agent is able to move code and data to a remote site and to adapt itself to the target run-time environment. Mobility includes remote execution and migration. Remote execution means moving data and code to a remote site which hosts the entire execution whilst migration means moving data and code during execution to perform progressively some tasks through several sites.

Nevertheless agents do have some basic characteristics :

- *autonomy* of an agent is twofold. First, an agent must be able to perform its tasks with a very loose coupling with others agents and users. Second, an agent must be able to initiate activities - like cognitive inference or cooperation/collaboration with other agents - on its own according to reached internal states and/or occurring events,
- *asynchronism* of behavior which means that triggering of the actions of an agent must be governed by its internal rules only. For instance, the invocation of a service provided by an agent can be delayed according to its current state and running activities,
- *communication* is the basis of the cooperation and collaboration between agents. An agent must be able to communicate with other agents which they know about the functionalities. It can also get in touch with other agents by trading mechanisms to know about their functionalities.

All these previous attributes make agents very relevant for the most and most complex needs in open distributed systems. Among these needs, a very difficult one to achieve is the slogan «*all information available at any time in any place*». The «*electronic market place*» metaphor is becoming a standard in the new electronic services. Services must be provided by electronic entities able to meet their consumers, achieve contracts with them and provide their services in a secure way. This need disclaims classical design and implementation solutions and asks for «*intelligent collaborative/cooperative mobile agents*».

3.2 Agents vs. Objects

Agent paradigm shares with object-oriented paradigm a basis of underlining concepts like *abstraction*, *modularity* and *encapsulation*. Agents can be viewed as elaborated entities which have capabilities such as reasoning and collaborating/ cooperating that are not represented in classical object-oriented paradigm.

Nevertheless, some major differences exist between the two paradigms and agents can not be restricted to «*thinking objects*». In object orientation, one can distinguish between active and passive objects while agents are inherently active and autonomous entities. We can also notice that objects are essentially deterministic and have sequential methods often invocable one at a time. In agent-orientation concurrency (e.g. reasoning while collaborating/cooperating) and indeterminism (e.g. execution depending on context agreed on in collaboration/cooperation) are necessary. Method invocation in object -orientation is often free and only constrained by type-checking of the parameters while agents might interact according to well-defined protocols which can make some invocations allowed or not at a given context.

However, object-oriented paradigm can be a valuable basis to build agent-based systems because they offer a first structuring level of the system into entities that can be extended to agents. The concepts like modularity, encapsulation, elaborated interfaces can be applied to agents in a satisfactory way as a first structuration level.

4 SERVICE IN OPEN DISTRIBUTED SYSTEMS

Agent paradigm is expected to emphasize in emerging open distributed systems the concept of *service* (TINA-Consortium 95, Merz 96). A service is a collection of functionalities provided to be used under some conditions (Merz 96, Diagne 96a). The functionalities are supported by information stored and manipulated for providing the service. This information models the *resources* necessary to run the service. The conditions attached to a service can be of many kinds. A service can be attached with a protocol which specifies how to use it correctly e.g. by sequencing in a given way its many functionalities. It can also be submitted to access permissions or quality aspects which may vary according to the context in which it is used. Finally it can be submitted to any other restriction relevant to be considered in its application domain e.g. taxation, temporary (un)availability, progressive adaptation to users needs and contexts, etc.

Each functionality of a service is implemented by an operation the environment can invoke. The view the environment has on an operation consists of its quality(ies) of service, its signature and constraints from the service view (dependencies with other operations). A service is offered by a server and can be requested by clients. At the server side, the operation is implemented by a given behavior which can be run with different strategies according to the expected quality of service. For instance the same *behavior* which broadcasts a message is run with different *strategies* in order to achieve a reliable broadcast or an unreliable one. The service must therefore implement the behavior as well as its different strategies of application.

The concept of service must take into account the many aspects ranging from reactive properties to elaborated cognitive and collaborative/cooperative capabilities. Specification, realization, test and deployment of services need to be considered in a methodical way. Formal methods are being more or less applied to fullfil the needs of safety and reliability (Gervais 96).

Services encompass many aspects like collaboration with its users, reasoning on contextual knowledge and executing some functionalities to achieve the service. These aspects can be divided into three parts:

- the *collaborative/cooperative aspects* like negotiating and making contracts with users or other services. These contracts can be commitments on quality of service and/or access rights, consequent billing, etc.
- the *cognitive aspects* like making inference on the contextual knowledge attached to a service or to its execution. This contextual knowledge can determine the way the service is offered. It can also be associated to the profile of the service user,
- the *reactive* or *computational aspects* like modifying the resources and running some specific processing necessary to exhibit the right functionalities under the contractual constraints. Services are namely reactive because they must maintain a continuous interaction with their environment (users or other services).

These previous aspects are not independent from each other. We can notice that the contextual knowledge used in cognitive activities may depend on the previous collaborative/cooperative activities and may influence the reactive ones. These interdependencies must be considered while using formal methods in order to support verification and validation. For instance, formal methods can be considered on reactive aspects to ensure safety and reliability whatever are the cognitive and collaborative/cooperative aspects (Estraillier 96). We can so aim more normalization in the agent-based open distributed systems (Pitt 95). It appears therefore necessary to separate and manage the many concerns in order to avoid undesirable influences from each other (Hursch 95).

5 ADAPTING VIEWPOINTS TO THE AGENT PARADIGM

As agent-based systems need more architectural guidelines to achieve a first level of integration as well as a good separation of concerns, we propose to proceed like in the RM-ODP with some adaptations (McKay 96). The ODP viewpoints are well-suited to separate the many concerns in object-based open distributed systems. Nevertheless, we would not propose viewpoints for agent-based systems to be a progressive structuration like in ODP. We try, through our adaptation of viewpoints, to separate and organize the many concerns (collaborative/cooperative, cognitive and reactive) of such systems and to make their mutual dependencies more manageable. Viewpoints help us to separate concerns. We propose to apply the concepts underlining the ODP structuration to the agent-based system and we propose the following classes of agents. *For that purpose and for the understandability of the remainder of the paper, the reader is warned of the fact that in this proposal, all en-*

tities are processing ones unlike in RM-ODP where enterprise and information objects are not processing ones.

5.1 Service Manager Agents

A Service Manager Agent is an entity managing the policies and rules attached to the availability and utilization of a collection of services. A Service Manager must be able to negotiate the offer of its services with consumers. It must also be able to cooperate with other Service Managers to use their services when needed. The two kinds of customer (humans and others Service Managers) put high requirements on the interface mechanism. High level trading capabilities and elaborated user interfaces are necessary in order to present services for the understandability of humans and availability for electronic entities.

A Service Manager must be able to represent the knowledge necessary to run the service and the relevant reasoning capabilities on that knowledge. Service Manager Agents can be mobile or fixed agents. They are responsible to negotiate with others agents or users in order to determine a context under which services will be provided (Finin 94). They match the collaborative/cooperative and cognitive aspects of a service.

Service Managers can cooperate between them to share the load of the service offer. Thus they do have distributed problem-solving capabilities to share their knowledge and the inference they perform on it (Carle 95). Load and knowledge sharing can be used by a set of Service Managers to present some kind of cooperation that supports *«shared state»* in the system.

Concerning the collaborative aspects, Service Managers must be able to establish contracts and to fullfil the subsequent obligations (Pitt 95). Service Managers must be able to accept or deny results of negotiation but once accepted, the subsequent contract must be carried out in a satisfactory way for the counterpart.

At this level, best effort must be put on separation of cognitive activities from the collaborative/cooperative ones. Collaboration and cooperation might need some cognition and cognition might depend on previous collaboration. The dependencies between the both aspects must be clearly identified and validated even in an informal way. The role of a Service Manager will be further clarified as we go along.

5.2 Resource Manager Agents

A Resource Manager Agent is an entity responsible to manage one or many resources on behalf of a Service Manager. It offers capabilities for access and modification of the managed resources. It defines the allowed access to the managed resources as well as integrity constraints that will be enforced.

Resource Managers are under control of a given Service Manager which can give - by authentication means - an access/modification permission to other entities (any other kind of agent) (Thirunavukkarasu 95). This permission will determine a subset of allowed accesses and modifications. Resource Manager can be made mobile in order to make the information available on remote site but the mobility is under control of the corresponding Service Manager. Resource Manager can encapsulate part of the reactive aspects of the service related with resource manipulation.

Exception processing must be enforced at this level to send some events back to the Service Managers because they might need some cognitive or collaborative actions. For instance, a temporary or definitive unavailability of some functionality(ies) must be signalled to the Service Manager to make it change its proposal to the environment relevantly. This allows to offer adaptable services. Access/modification attempts by agents which are not truthful or without the right permissions need also to be signalled back to Service Managers. These events are trapped by the Service Managers like exceptions, so the relevant activities can be performed. Exceptions allow one to have some kind of fault tolerance on Resource Managers.

Some Resource Manager may need to access functionalities offered by a counterpart in order to achieve its own ones. It must therefore ask its responsible Service Manager to undertake the necessary collaborative and cognitive activities in order to obtain the access/modification permission. This permission is delegated to the Resource Manager which therefore can send invocations. This problem can be solved by permanent contracts between Service Managers in order to access remote resources transparently. The access to the remote resource by the local Resource Manager can be achieved by creating ad-hoc Activity Managers (see next section).

We make the choice that Resource Managers do not receive any collaborative/cooperative or cognitive capabilities from their responsible Service Managers. They are therefore pure reactive agents which run according to the access permission attached to received invocations. We can then validate them formally and make sure that resources will not be corrupted independently from the indeterminism in collaboration/cooperation and cognition. Resources are critical to the correct operating of the service and therefore, all their concerns must be validated and verified formally.

5.3 Activity Manager Agents

An Activity Manager Agent is an entity able to perform a set of actions in order to fullfil a given goal. An Activity Manager is under control of a Service Manager and receives from it access permissions on its attached Resource Managers. It can also - according to its fixed goal - receive collaborative/cooperative and cognitive capabilities to address others Service Managers. Another possibility which seems best is that the responsible Service Manager carry on the collaborative/cooperative and cognitive activities with other Service Managers, then the Activity Manager only receives delegation on permission granted to its responsible Service Managers to access remote resources.

Activity Managers may divide their goals into sub-goals. Therefore, they clone themselves into others Activity Managers to handle these sub-goals. The global coherence must then be managed by the agent which initiates the goal splitting. Transaction-oriented facilities must be supported to manage this coherence. Activity Managers can have collaborative, cognitive and reactive aspects according to their assigned goal. They are anyway attached to a given Service Manager which will delegate them part or whole of its collaborative/cooperative and cognitive capabilities.

Activity Managers realize the reactive tasks necessary to Service Managers. So at the level of Service Managers, we can only consider collaborative and cognitive aspects. This delegation is a way to isolate the reactive aspects from the others in order to formally verify their safety and reliability. Like for Resource Managers, we need to trap some exceptions for the behalf of Service Managers in order to process exceptional events that might happen to Activity Managers. For instance, a definitive failure on a (sub-)goal can eventually need new cognitive and collaborative/cooperative activities which the Activity Manager can not carry out on its own.

5.4 Engineering and Technological Agents

The two last viewpoints in the RM-ODP deal mostly with implementation aspects. In the agent paradigm also, we will consider such kind of agents as relevant to realize the three previous classes we have defined. Depending on an underlining technology, one must consider how the previous levels of abstraction can be realized. What can be called *engineering* and *technological agents* must therefore be defined to establish some correspondence between the needs in the previous levels with the concepts available in the underlining technology. We remain deliberately unprescriptive and refer to the RM-ODP for adaptation.

6 Service-based Architecture of an Agent-based System

Agent and Service are two valuable concepts to structure open distributed systems. They offer two levels of structuration which can be mixed to have a federative basis for such systems. Services can stand for structuration unit for which agents are used to represent their many concerns with a clear separation between them.

6.1 Overall Architecture

Given the set of agent classes we define above, we are going to issue an architectural proposal for structuring open distributed systems according to service and agent concepts. We will consider here the key idea of service as a main guideline in structuration of systems. We consider henceforth that a system is characterized by the set of services it can provide to its environment.

A service can be structured as follows:

- one or many Service Manager Agents responsible of the *policy of the service* encompassing its *use* by the environment and its eventual *collaboration* and *cooperation* with other services. They have the same lifetime than the service,
- one or many Resource Manager Agents responsible of the *local resources* necessary to the service and which it owns and manages. Their lifetime is up to the needs of the responsible Service Manager Agents,
- one or many Activity Manager Agents whose *goals* are determined by some Service Manager Agent. Then, they receive from that Service Manager eventual collaborative/cooperative and cognitive capabilities necessary to fulfill that goal. Their lifetime can end with the definitive success or failure for the as-

signed goal.

The relevancy of the service notion here is its reflexivity. In complex services, one can undertake decomposition and each Service Manager can be considered - with its attached Resource Managers and Activity Managers - as an elementary service. This decomposition allows to consider services at a granularity level which does not carry too much complexity. So test validation and verification can be achieved in a satisfactory way on elementary services before combining them into more elaborate ones. The feature interaction problematic deals with problems that might occurs like degradation of service functionalities caused by concurrency between instances of many services (or instances of the same service) is well known in telecommunication systems (Cameron 94). It can be tackled better at this low-level of granularity and it must be taken into account in the services composition procedure.

The delegation of reactive tasks to Activity Managers by Service Managers can be done one the fly and when needed. The Activity Managers do not need in that case any more collaborative or cognitive aspects. They are only created to perform a given task on behalf on an Service Manager. The Activity Managers can then be validated using formal methods in their reactive aspects. This possibility is left up to system designer and must be evaluated in front of the level of validation one can need in a given application domain.

6.2 Synthesis and Discussion

The ODP viewpoints are used to separate concerns in systems while managing consistency between them. We adapt them here in order to manage the dependencies between different aspects of a system (a collection of services). The progressiveness from Enterprise viewpoint to Technological one is lost because we have more operational associations to manage between a Service Manager and its attached Resource Managers and Activity Managers. Viewpoints are used to assign roles and activities to components in a service.

In figure 1 we show the correspondence between the ODP separation of concerns and the one we propose in this paper in order to structure services in an open distributed system by means of agents. Service Managers model the adaptation of the Enterprise viewpoint. Service Manager Agents need to represent the policy and general rules which govern service availability. They use these policy and rules to make contracts with users or other Service Manager Agents. Their representation of resources and activities is achieved through their attached Resource Manager and Activity Manager Agents. Resource Managers correspond to the Information viewpoint. Resources Manager Agents only represent the schemas that govern the management of resources. They can represent some computational activities in order to be able to invoke other local Resource Manager Agent or remote ones based on pre-established contracts between their akin Service Manager Agents. Activity Managers are the most complex entities because they must be flexible enough to allow the system designer to model different strategies. So they have basically computational aspects but can be enhanced with collaborative/cooperative and cognitive aspects as well as management of some temporary resources necessary for one execution at most This kind of resource, like the contextual information necessary to continue execution on remote sites in case of migration, does not enforce the need of a Resource Manager. Therefore, they can have capabilities of Resource Manager Agents and also receive delegation of e.g negotiation capabilities from their responsible Service Manager Agents. This allows them to make contracts with other Service Manager Agents.

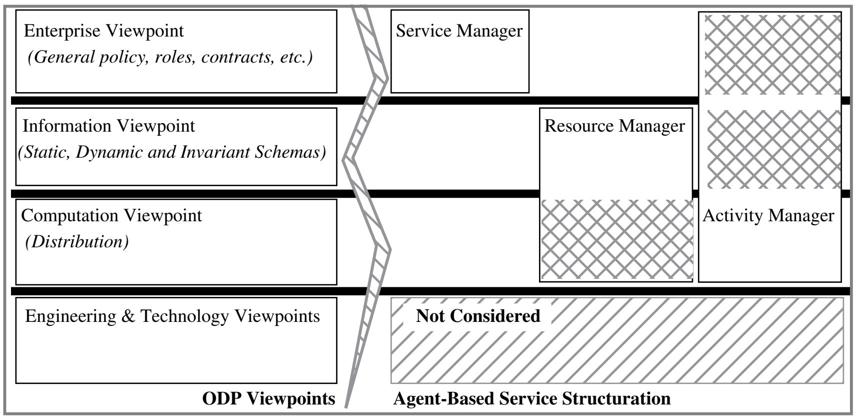

Figure 1 From ODP Viewpoints to Agent-based Service Structuration

An obvious advantage of this proposal is the fact that one can isolate the information managed (resources) in the Resource Manager Agents which can be validated formally. We therefore make sure that whatever the collaborative/cooperative and cognitive activities are, the resources will not be corrupted. Resources are critical for a service because their states determine the correctness, safeness and reliability of its behavior. For these reasons, it is important to verify and validate thoroughly their concerns.

7 CONCLUSION AND FUTURE WORK

We propose in this paper a way to mix the agent and service concepts and enhance them with architectural concepts which can fit for all purposes from the specification stage down to the final implementation. Agent-based service structuration of open distributed systems can therefore be emphasized with a satisfactory level of separation of concerns.

The proposed architecture first attempts a separation of concerns in order to better manage their own semantics and mutual dependencies. It also enables a service designer to apply formal methods when and where they can be needed to improve safety and reliability verification. This separation of concerns is difficult to achieve fully in some architecture because it depends on the intention and will of the service designer and the constraints of application domain. For instance the decision to make Activity Managers fully reactive in our proposal is up to the service designer under constraints enforced by the application domain.

In case of indeterminism in goals assigned to Activity Managers, it will be difficult to prevent them from having collaborative and/or cognitive capabilities in order to maintain a loose coupling with their responsible Service Managers. However, we give a valuable way to realize it more or less. We also claim that the architecture can be a first level of integration in the «*open electronic market*». Services designed in a similar way are more easy to make interwork than otherwise. We have defined in (Diagne 96a) and (Diagne 96b) models which fit for the reactive aspects of our architecture proposal. We will henceforth stress our research effort on collaborative/ cooperative and cognitive models.

8 REFERENCES

H. Bowman, E.A. Boiten, J. Derrick & M.W.A. Steen, «*Viewpoint Consistency in ODP, a General Interpretation*», In Proc. of FMOODS'96, Paris, France, March 1996.

E.J. Cameron, N.D. Griffith, Y.-J. Lin, M.E. Nelson, W.K. Shnure & H. Velthuijsen, «*A Feature Interaction Benchmark in IN and Beyond*», In Feature Interactions in Telecommunications Systems, IOS Press, Amsterdam, Holland, 1994.

P. Carle, A. Collinot & K. Zeghal, «*Cassiopeia : a Method for Designing Computational Organizations*», In Proc. of IJCAI'95, Montreal, Canada, Aug. 1995.

A. Diagne & P. Estraillier, «*Formal Specification and Design of Distributed Systems*» In Proc. of FMOODS'96, Paris, France, March 1996.

A. Diagne & F. Kordon, «*A Multi-Formalism Prototyping Approach from Conceptual Description to Implementation of Distributed Systems*», In Proc. of the 7th IEEE Int. Workshop on Rapid System Prototyping, Greece, Porto Caras, June 1996.

P. Estraillier & F. Kordon, «*Structuration of Large Scale Petri Nets: An Association with High level Formalisms for the Design of Multi-Agent Systems*», In Proc. of the IEEE Int. Conf. on System Man and Cybernetics, Beijing, China, Oct. 1996.

J. Ferber, «*Les Systèmes multi-agents : Vers une intelligence collective*» iia interEditions, 1995.

M.P. Gervais & A. Diagne, «*Formalization of Service Creation in Intelligent Network*» In Proc. of the Int. Conf. on Intelligent Network, Bordeaux, France, Dec. 1996.

T. Finin, Y. Labrou & J. Mayfield, «*KQML as an Agent Communication language*», In Proceedings of ACM/CKIM'94, ACM Press Nov. 1994.

W.L. Hursch & C.V. Lopes, «*Separation of Concerns*», Tech. Rep. NU-CCS-95-03, College of Computer Science, Northeastern University, Boston, USA, Feb. 1995.

T. Magedanz, K. Rothermel & S. Krause, «*Intelligent Agents : An Emerging Technology For Next Generation Telecommunications?*», in Proc. of IEEE/INFOCOM'96, San Francisco, USA, March 1996.

M. Merz & W. Lamersdorf, «*Agents, Services and Electronic Markets : How do they integrate ?*», In Proc. of the Int. Conf. on Dist. Platforms, Dresden, Germany, 1996.

D.P. McKay, J. Pastor, R. McEntire & T. Finin, «*An Architecture for Information Agents*», In Advanced Planning Technology, AAAI Press, Menlo Park, CA, USA, May 1996.

J. Pitt, M. Anderton & J. Cunningham, «*Normalized Interactions Between Autonomous Agents : A Case Study in Inter-Organizational Project Management*», In Proc. of COOP'95, Antibes-Juan-Les-Pins, France, Jan. 1995.

C. Thirunavukkarasu, T. Finin & J. Mayfield, «*Secret Agents - A Security Architecture for KQML*», In Proc. of ACM/CKIM'95, Agent Workshop, Baltimore USA, Dec. 1995.

ITU X.902 & ISO/IEC 10746-2, «*Basic Reference Model of Open Distributed Processing, Part 2: Descriptive Model», 1995.*

ITU X.903 & ISO/IEC 10746-3, «*Basic Reference Model of Open Distributed Processing, Part 3: Prescriptive Model», 1995.*

TINA-Consortium, «*Service Architecture*», TB_MDC.012_2.0_94, 31 March 1995.

Java, Sun Microsystems, «*The Java Language Environment : a White Paper*», http://javasoft.com/, 1995.

Telescript, General Magic, «*Telescript Language Reference*», http://www.genmagic.com/, 1995.