Designing Peer-to-Peer Applications: an Agent-Oriented Approach

D. Bertolini[1], P. Busetta[1], A. Molani[2], M. Nori[1], and A. Perini[1]

[1] ITC-Irst, Via Sommarive, 18, I-38050 Trento-Povo, Italy
{bertolin,busetta,nori,perini}@irst.itc.it - phone +39 0461 314 330
[2] University of Trento, via Sommarive 14, I-38050 Trento-Povo, Italy
molani@science.unitn.it

Abstract. This paper focuses on design issues to be faced when developing knowledge management (KM) applications based on the integration of peer-to-peer and multi-agent technologies. The reasons for using these technologies rest on the requirements posed by the specific KM paradigm that has been adopted, which emphasizes aspects such as *autonomy* and *distribution* of knowledge sources. We adopt an agent-oriented approach that extends *Tropos*, a software engineering methodology introduced in earlier papers. We present a characterization of peer-to-peer in terms of a general architectural pattern, a set of design guidelines for peer-to-peer applications, and a framework that integrates multi-agent and peer-to-peer concepts and technologies.

1 Introduction

Emerging application areas such as *distributed knowledge management* (DKM) [2] call for open as well as robust and secure software systems. Several technological solutions – sometimes combined – have been proposed to address some of these non-functional requirements, including e-services, application service provision, multi-agent systems (MAS), and peer-to-peer architectures (P2P).

A question naturally arises: how can we compare these technologies and evaluate which one is better given an application's requirements, or decide if they can be integrated in our application? For instance, some P2P concepts and technologies – such as *peer* and *peer group* – are well suited to capture and support certain aspects, such as the dynamism of the social organizations using a distributed system; typical MAS concepts – such as *agent* and *multi-agent systems* – are better suited to capture reasoning and collaboration among distributed components; e-services and ASPs promise to be widely available and to become important technological players in the near future, in spite of their current clumsiness in supporting real openness and robustness.

We present here some results of our work on a development framework for distributed knowledge management applications, which attempt to give some answers to the question raised above. In particular, we focus on the problem-solving process underlying the design of DKM applications. Objective of this process is to specify how to implement the system's functional requirements while respecting

the constraints imposed by the non-functional requirements (NFR). We adopt an agent-oriented approach by extending a software engineering methodology, called *Tropos*, presented in earlier papers [5].

The main contributions of this work are three. The first is a characterization of peer-to-peer in terms of an architectural pattern, obtained by applying the *Tropos* methodology. The second is a set of guidelines for a system designer on when and how to apply a P2P approach; these guidelines are targeted at a specific class of applications (DKM) but can be easily generalized to others. The third result is an architectural framework that integrates P2P and multi-agent systems, basically proposing P2P as a way of deploying an open MAS and providing some supporting services, most notably discovery.

This paper is structured as follows. Section 2 briefly presents the distributed knowledge management approach that motivated this research, and other work relevant to this paper. Section 3 introduces some basic Tropos concepts and notations, which are used, in Section 4, to discuss some typical peer-to-peer architectures and to identify their common patterns. In Section 5 we focus on JXTA, the P2P framework we are currently using. Section 6 presents the guidelines for designing the architecture of a DKM system applying both P2P and MAS concepts. Section 7 describes a scenario taken from the Health Care domain, and shows how to apply the guidelines in practice. Finally, conclusions and future works are presented in Section 8.

2 Background and Related Work

Common knowledge management (KM) systems support the collection and categorization of knowledge with respect to a single, shared perspective, and its redistribution to its users by a variety of means. In many instances, this leads to the construction of one or a few repositories of documents, organized around a single ontology or other meta-structures. Users are given tools to search, browse, or receive documents as soon as they become available, varying from simple Web interfaces to sophisticated agents running on the users' desktops.

However, common wisdom is that building the shared perspective at the core of a KM systems is an expensive task, sometimes impossible to achieve when users have substantially different goals and background. To tackle this issue, our research group is investigating a novel approach called *distributed knowledge management* (DKM) [2]. The idea at the basis of DKM is supporting the integration of autonomously managed KM systems, without forcing the creation of centralized repositories, indexes, or shared ontologies. This integration involves the deployment of a set of complex protocols and algorithms for information retrieval, natural language processing, machine learning, deductive and inductive reasoning, and so on, sometimes requiring the direct participation of human users. Initial publications include [8, 3].

To build such systems, a clear understanding of how available technologies can be combined to deal with functional and non-functional requirements becomes a critical issue. Software engineering provides the designer with a set of

principles aimed at decomposing the system design problem into subproblems and at favoring the re-use of previous solutions. We are considering these issues from an agent-oriented perspective, by extending the *Tropos* software engineering methodology.

The extensions to Tropos for the support of DKM are part of the development of a larger framework, whose aim is to provide both a methodology and technological support to analysts, designers, and developers. As part of this framework, we are investigating both multi-agent and peer-to-peer technologies.

Different lines of research are relevant to the work presented here, ranging from peer-to-peer and agents applications, to agent oriented software engineering in general [15]. For instance, two recent works have attempted to blend peer-to-peer and agents into a single framework: Anthill [1] and InfoQuilt [11]. Both see peer-to-peer as a paradigm for networks of autonomous systems that may join or leave at any time. Relevant to our work are also studies on peer-to-peer systems that have been conducted using i^* [16] (the modeling language at the basis of Tropos), devoted to the characterization of the base technology with respect to critical issues such as security and neighborhood analysis [7, 12].

3 An Overview of Tropos

The *Tropos* methodology [5] adopts an agent oriented approach to software development starting from the very early stage of requirement specifications, when the environment and the system-to-be are analyzed, down to system design and implementation. The methodology identifies a number of phases (*early requirements*, *late requirements*, *architectural design*, *detailed design*, and *implementation*) and has been applied to several case studies [6]. The core process of the methodology consists in performing conceptual modeling using a visual language that provides intentional and social concepts such as actor, goal, belief, plan and dependency. An *actor* models an entity that has strategic goals and intentionality, such as a physical *agent*, a *role* or a *position*. A *role* is an abstract characterization of the behavior of an actor within some specialized context, while a *position* represents a set of roles, typically covered by one agent. The notion of actor in Tropos is a generalization of the classical AI notion of software agent. *Goals* represent the strategic interests of actors. A *dependency* between two actors indicates that an actor depends on another in order to achieve a goal, execute a plan, or exploit a resource. Tropos distinguishes between hard and soft goals, the latter having no clear-cut definition and/or criteria as to whether they are satisfied. Softgoals are useful for modeling software qualities [4], such as security, performance and maintainability. A Tropos model is represented as a set of diagrams. In particular, actor and dependency models are graphically represented as *actor diagrams* in which actors are depicted as circles, their goals as ovals. The network of dependency relationships among actors are depicted as two arrowed lines connected by a graphical symbol varying according to the dependum: a goal, a plan or a resource. Actor goals and plans can be analyzed from the point of view of the individual actor using three basic reasoning tech-

niques: *means-end analysis*, *contribution analysis*, and *AND/OR decomposition*. During this analysis, new actor dependencies can be identified.

Tropos is currently being extended with concepts suitable to model some specific notions – such as distributed knowledge, autonomy and coordination – that are peculiar to DKM, in order to support an early requirement model of a DKM domain [9].

In this paper, we focus on a later phase in software development: given a requirement analysis of a DKM problem, we provide guidelines for defining one or more suitable system architectures. According to *Tropos*, these types of issues are typically faced during the *architectural design* phase.

4 The Peer-to-Peer Virtual Community Pattern

In this section, we use Tropos to analyze which functional and non-functional requirements are successfully satisfied by peer-to-peer systems [10]. In particular, we consider two well known systems, Gnutella and Napster, both supporting file sharing (in particular MP3 sound files).

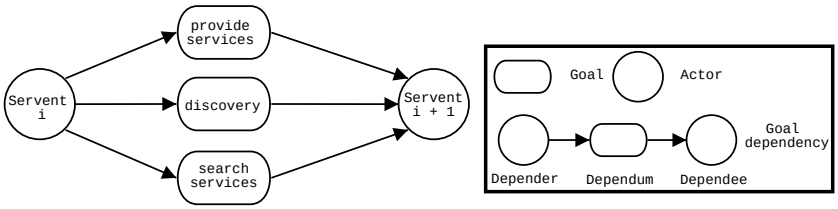


Fig. 1. Gnutella: architectural design, according to the Tropos methodology

The logical architecture of *Gnutella*[3] is described in the *Tropos* actor diagram depicted in Figure 1. Both actors model a basic role, the Servent (Gnutella's terminology for peer). Each servent has the goals of discovering other servents, looking for MP3 files, and providing its own MP3s, and depends on all other servents for achieving them. In other words, the goals of a servent are achieved only by means of a virtual community of peers, each one playing the role of servent. In the diagram, this is represented as goal dependencies between two generic servents. The dependencies are symmetric for every servent; for simplicity, we have shown them going only in one direction. The discovery goal enables the community to be dynamic, since peers can join and leave the community at any time without having to register their presence anywhere. There is no centralized service local to any one peer; conversely, all the peers provide (and take advantage of) the same sets of services.

[3] `http://www.gnutella.com`

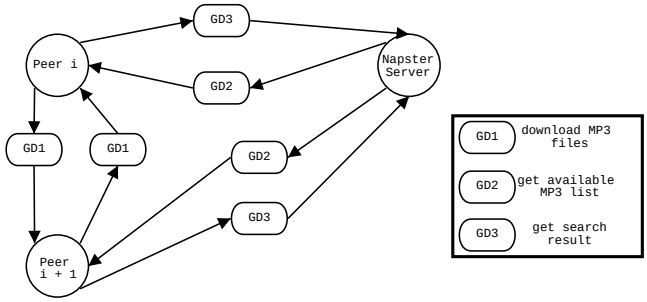


Fig. 2. Napster: architectural design

In *Napster*[4], each client, represented by Peer actors in Figure 2, has similar goals to those of Servents in Gnutella (search and download files). However, in order to obtain a search result, the client must contact a central server (Napster Server). This server maintains a list of all the active clients and a list of all the shared MP3 files. The server depends on the clients for building these lists. This situation is represented by a set of goal dependencies in Figure 2: the generic Peer depends on Napster Server for getting a search result (goal dependency GD3) and Napster Server depends on Peer to get the lists of available MP3 files (goal dependency GD2). Analogously to Gnutella, two generic peers depend on each other for downloading MP3 files (goal dependency GD1). So, the individual goal is obtained by means of a community of peers (Napster clients) that coordinate with an actor playing a distinct role in the community, i.e. the server.

We abstract these two architectures, and others not discussed here, in the basic model depicted in Figure 3. The Individual actor models somebody who has at least one of two goals: accessing a resource (or, equivalently, using a service); and, letting others access her own resources (or use her own services). An individual has a set of constraints and preferences (shown as softgoals in the diagram) which drive her behavior when achieving these goals. In particular, an important requirement is being autonomous, i.e. being able to control what to access or to make available to the external world, and when.

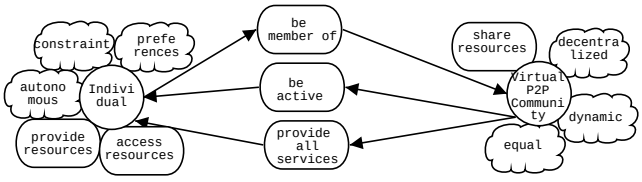


Fig. 3. The peer-to-peer virtual community pattern

[4] `http://opennap.sourceforge.net`

The actor Virtual P2P community has the main goal of letting its members cooperate for sharing a specific type of resource or service. Three additional requirements (shown as softgoals) have been identified: being highly dynamic – available resources and services provided by members can be added or removed at any time –; being decentralized, i.e. the community is able to achieve its main goal independently of any specific member or component; and finally, being composed of peers on an equal basis, that is, every member is compelled to provide (at least potentially) resources or services and has the right to access the services of others.

The dependency be member of captures the fact that the individual's goals can be satisfied by joining the community. The act of joining implies the acceptance of the community's main goal and rules, highlighted above. Conversely, a community can achieve its goal only if it is active, and this can be accomplished only by having individuals as members. Finally, the dependency provide all services models the rule that in a peer-to-peer community all members are equal, i.e. provide the same set of services.

Going back to the systems discussed previously, Gnutella is a "perfect" P2P system, since it satisfies all non-functional requirements of the community highlighted in Figure 3. Similarly, it may be argued that Napster is not a real P2P system, since the community depends on a centralized service.

5 JXTA P2P: Logical Architecture Elements

We focus now on *JXTA*[5], a set of open, generalized peer-to-peer protocols that allow devices to communicate and collaborate through a connecting network.

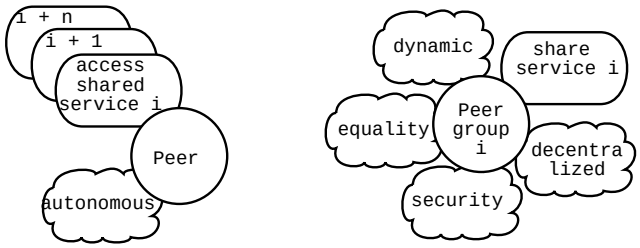


Fig. 4. JXTA: architectural design overview

From our perspective, the fundamental concepts of JXTA are three: *peer*, *peer group* and *service*.

A *peer* is "any device that runs some or all the Project JXTA protocols". The complex layering of the JXTA protocols and the ability for a peer to simul-

[5] A P2P open source effort started in April 2001. `http://www.jxta.org/` and `http://spec.jxta.org/v1.0/docbook/JXTAProtocol.html`

taneously participate to more than one peer group (described below) imply that a peer is – at least conceptually – a multi-threaded program.

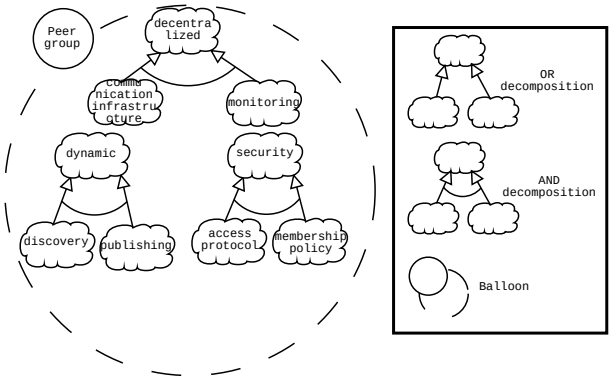


Fig. 5. JXTA: PeerGroup architectural design

A *peer group* is a collection of peers that have agreed upon a common set of rules to publish, share and access "codats" (shorthand for code, data, applications), and communicate among themselves. Each peer group can establish its own membership policy, from open (anybody can join) to highly secure and protected (join only if you have sufficient credential). Thus, a peer group is used to support structuring (based on social organizations or other criteria) on top of the basic peer-to-peer network. As mentioned above, a peer may be part of many groups simultaneously.

A peer provides to each of its group a set of *services*, which are advertised to the other members of the group. In other words, groups in JXTA support a model for service publishing which is alternative to the traditional, centralized directory services model. Each group specifies a set of services that have to be provided by its members; they may include supporting basic JXTA protocols (e.g. discovery) as well as user-written applications, such as file sharing.

From a high-level perspective, these concepts can be modeled in terms of roles in the architectural design diagram depicted in Figure 4, which extends the pattern depicted in Figure 3. A Peer has n goals access shared service i, one for each type of service it needs, and the requirement (specified as a softgoal) of being autonomous. A generic actor PeerGroup has the goal of sharing a service of a specific type i and a set of requirements: to support the decentralized distribution of services, to allow for dynamic membership of the group, to support security management, and to require that all members provide equal services. JXTA tackles these requirements using a number of mechanisms, represented as goal decompositions in Figure 5. The publishing and discovery mechanisms, together with a message-based communication infrastructure and peer monitoring services, support decentralization and dynamism. security is supported by a membership

policy (which authenticates any peer applying to a peer group) and an access protocol (for authorization control).

In summary, it can be said that the creation of a PeerGroup follows from the need to define a set of peers that are able to communicate and interact with a common set of capabilities and protocols, and that are aggregated for security reasons (i.e., no extraneous peer can intervene amongst them).

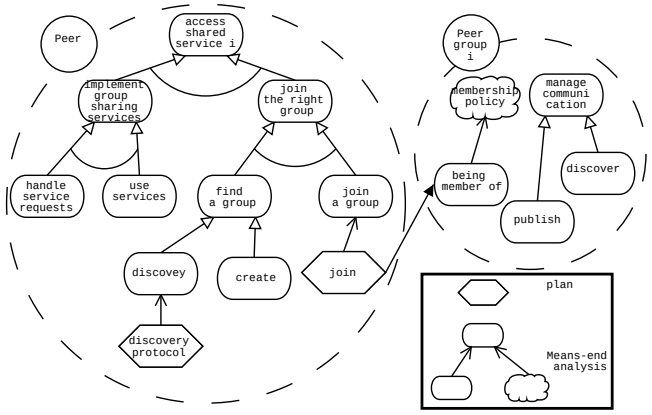


Fig. 6. JXTA: Peer architectural design

Figure 6 shows that a Peer's main goal (access shared service i) can be decomposed into two subgoals: join the right group and implement group sharing services. The first means that the peer must find a group that it is authorized to join and that provides the services it needs. This goal can be further decomposed into join a group and find a group. The latter can be satisfied by using the JXTA discovery service; if no adequate group is found, a new one can be created. Joining a group that has been discovered or created depends on its membership policy. Once a group has been joined, a Peer must implement all the services required by that group, implementing both the client side (use services) and the server side (handle service requests). It is important to stress that a Peer is autonomous in deciding how to provide a service (only protocols are predefined), and that a Peer can join different PeerGroups in order to achieve different goals.

6 Architectural Design of Communities in a DKM System

In this section, we sketch the guidelines for designing the architecture of a DKM system, focusing on the support for virtual communities. Input to the process described here is, in Tropos terms, the output of a late requirements phase (Section 3); that is, a domain analysis which included the system-to-be. The late requirements phase led to the identification of the stakeholders, including

knowledge and service sources and of their users; how this analysis is performed is outside the scope of this paper. The guidelines consist of the following steps:

1. apply the P2P virtual community pattern;
2. design the JXTA peer group implementing the virtual community;
3. design the agents that implement a peer service.

Each step is briefly described below, while in the next we section present their practical application.

Applying the P2P Virtual Community Pattern At this step, the designer has to answer to a basic question: can a Tropos late requirement model of a DKM problem be refined as a virtual P2P community model?

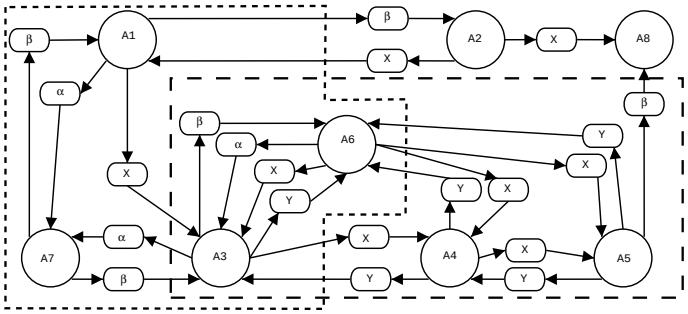


Fig. 7. Identifying P2P virtual communities. The *first community* (dashed line) is composed by actors involved in pair dependencies for *goals X and Y*, the *second* (dotted lines) by actors involved in pair dependencies for *goals* α *and* β

By definition, actors in a DKM scenario, as well as individuals in the community pattern, want to keep their autonomy. A designer, then, should look for the following conditions to be realized (they are illustrated in Figure 7 where two P2P virtual communities can be identified):

- there is a subset C of the actors that have pair-wise dependencies (e.g., actor A1 depends on A7 for goal α and A7 depends on A1 for β) in such a way that any actor is dependent on many, if not most, other actors in C. In other words, each actor is at the center of a sort of "star" network of reciprocal dependencies with many others;
- these dependencies can be generalized to one or a few types only. In particular, they are of type "access" and "provide" a service or a resource.

If these conditions are satisfied, then the designer can apply the P2P virtual community pattern to generate a model for a community supporting C. The main goals of this virtual community are suitable abstractions of the generalized dependencies linking the actors in C. The goals of an individual are also generated from each generalized dependency, and are two: achieve the objective of

the dependency, and, conversely, satisfy it. A validation step is necessary, and consists of two main activities:

- verifying that the goals of the actors in C can be effectively satisfied by adopting the goals of the individual in the community model; that is, verify that the P2P virtual community helps in achieving the actors' goals. This may be performed as a means-end analysis, which decomposes those hard goals into plans that, eventually, are satisfied by adopting the individual's goals;
- verifying that the general soft goals of a P2P virtual community – dynamism, equality, and decentralization – are at least partially achieved by the community being designed.

Designing a JXTA Peer Group Once that one or more P2P virtual communities have been identified, their supporting infrastructure can be designed. Assuming that JXTA is adopted as the basic communication technology, our major – and natural – choice is then to associate a JXTA peer to each individual, and a JXTA peer group to each P2P virtual community. Thus, a social actor (no matter if it is a single person or a group) will have, as its supporting system, a peer participating to as many peer groups as communities that have been identified during the requirement analysis of its DKM problems.

Agents as JXTA Services The final step is to design how individuals implement their goals with respect to each of the communities they form part of. From a JXTA perspective, it is necessary to specify which application services have to be provided by a peer, and how they are published on the network. JXTA, however, leaves the designer total freedom concerning their communication protocols and internal implementation. JXTA provides its own communication mechanisms (unreliable message queues called "pipes" and an XML based encoding format), but the application may decide to use something different, since JXTA supports publishing and discovery of communication end-points of any type (i.e., the address to which to send messages in order to obtain a service from a certain peer) as long as they can be represented in XML. Moreover, JXTA allows the publishing of additional service-specific information – also encoded as XML documents – along with their communication end-points; this gives a nice opportunity for targeted discoveries and filtering of potential peers.

The services to be provided by the peer associated to an individual are easily identified from the individual's goals with respect to the community. Once the peer protocols and publishing information have been defined, the design of the actors participating to the community can be performed in parallel. The Tropos methodology naturally leads to the design of multi-agent systems; thus, it is most likely that the implementation of services are agents themselves, interacting both with agents internal to their own actor and with other agents implementing other actors of the same community.

7 A Case Study

We show a practical application of the architectural guidelines given above to a DKM problem, taken from the Health Care (HC) domain. Health care is interesting both from a multi-agent perspective – as pointed out in [14, 17], where an approach is proposed to construct an information system centered on the individual user (customer-patient), instead of on the providers of information and services – and also from a knowledge management perspective [13]. Our scenario consists of the hospital wards involved in a specific patient treatment along with other related stakeholders. In our example, a patient with cancer may be admitted and pharmacologically treated by the oncological ward, operated by the surgical ward, and sent to the X-ray lab for analysis. All these stages may happen at different times, and possibly for different reasons. Each ward has its own local knowledge and a specific way of organizing and managing activities, processes, and information, according to its own primary objective. We assume

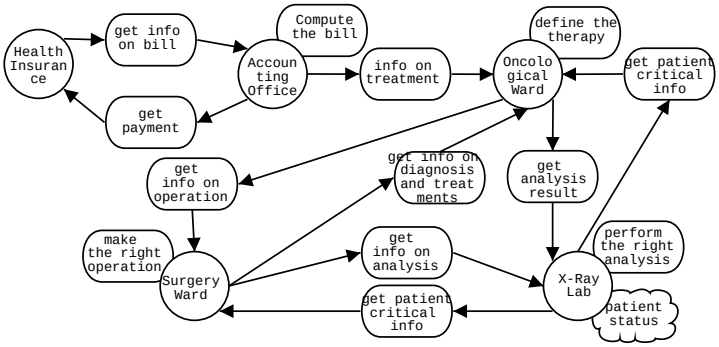


Fig. 8. The hospital wards scenario: an actor diagram

that the output of a deep requirement analysis led to the model depicted in Figure 8, where the actors represent the knowledge management systems of the entities involved in the scenario. The main goal of each actor has been delegated to it by its local users (not shown in the diagram). For instance, the Oncological Ward has the goal define the therapy. In order to achieve this goal, the Oncological Ward has to collect information on the surgery that the patient was subjected to and on possible post operation events and treatments. This is represented by the goal dependency get info on operation between the actor Oncological Ward and the actor Surgery Ward. Analogously, the Oncological Ward needs information also on the MRI or x-ray analysis performed by the X-Ray Lab, represented by the goal dependency get analysis result. Conversely, the X-Ray Lab depends on the Oncological Ward to get critical information about the patient. We represent additional requirements as softgoals; for example, X-Ray Lab's perform the right analysis goal is influenced by the patient status (e.g., a patient could be pregnant).

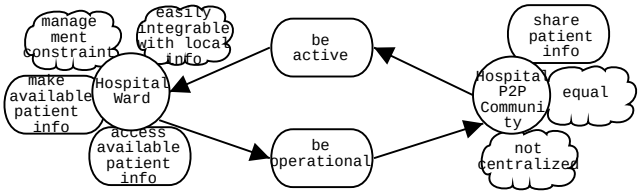


Fig. 9. Applying the P2P virtual community pattern.

To realize their mission, hospital wards need to cooperate, as highlighted by the goal dependencies among their KM systems. The result of this collaboration is a *distributed and virtual* clinical patient report. Privacy policy is a sensitive issue in this context. Following the first step of the guidelines presented in Section 6, we analyze the model depicted in Figure 8 and apply the virtual P2P community pattern (Figure 9). The result is a model where the community has the goal share patient info, and each individual has the generic goals of finding patient record information (access available patient info) and making available patient record information (make available patient info). It can be easily seen that these generic goals do actually contribute to achieving the wards' and labs' original goals. The community is not centralized, since all members keep their own information (as noted above, the members of this group have a strong interests in maintaining their autonomy), and all of them are on an equal basis because they need both to access and to provide information. The generic goal dependency, be member of, between an individual and a P2P community described in Figure 3 has been specialized as be operational. Dynamism of the community is probably minimal in this setting, since the number of wards and labs tend to be stable over time; however, new actors may join over time, for instance new wards, or mobile emergency teams. One of the advantages of designing a virtual P2P community is that, *after* the system's deployment, new actors can enter the community as long as their goals can be reformulated as the individual's goals. Figure 10 represents the output of the second step of our guidelines: a model for the peer group supporting the community identified above, according to the model of a JXTA peer group depicted in Figure 4. A custom security policy is specified, while all other other goals are adequately supported by the standard JXTA protocols and do not need to be analyzed any further. The membership policy allows only trusted peers to join the community; this control could be performed by using certificates and digital signatures, maintaining a list of trusted or revoked certificates, and granting particular credentials to every peer. Each time a peer uses a service, the access protocol checks the credentials of the peer in order to verify if it is authorized to access that particular service (not shown in the diagram). The final result of this second step is the definition of an environment – the peer group – that supports the joining, leaving and dynamic discovery of members, while at the same time providing a known level of reliability in terms of security and tolerance to failures of single components.

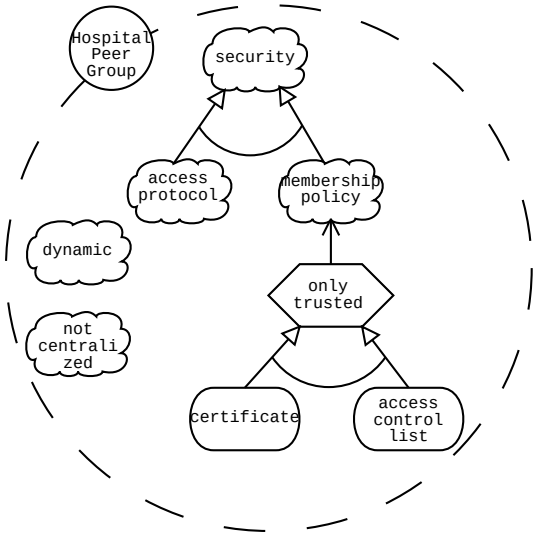


Fig. 10. The HC PeerGroup.

We finally apply the third step of our guidelines. The service provided by the peers of our community is an interface to their local databases containing patient information. We do not elaborate here on published information and protocols, which can be inspired by works on information integration systems, while we present a plausible architecture for a specific actor, the X-Ray Lab. Figure 11 contains only what is relevant to this discussion. The actor has a goal of its own (perform the right analysis), which, after a decomposition (not shown here), includes the sub-goal get patient info. This is delegated to a specific agent, the X-Ray Assistant. After further decomposition, the goal is achieved by accessing the community (access remote patient info, which is a reformulation of the generic individual's access available patient info of Figure 9).

Participating to the community implies that the X-Ray Lab has to satisfy the individual's make available patient info goal. This is delegated to a second agent, the X-ray db server, which operates both for the community and for local agents (see the access local db goal delegated by the X-Ray Assistant).

8 Conclusions and Future Works

This paper sketches initial results of a study aimed at defining a framework for building DKM applications, covering all development phases. The process is based on *Tropos*, an agent oriented software engineering methodology. We focused on the architectural design phase, when the application requirements are analyzed in the light of known architectural patterns, and defined some guidelines driving the development of DKM systems as a combination of a specific peer-

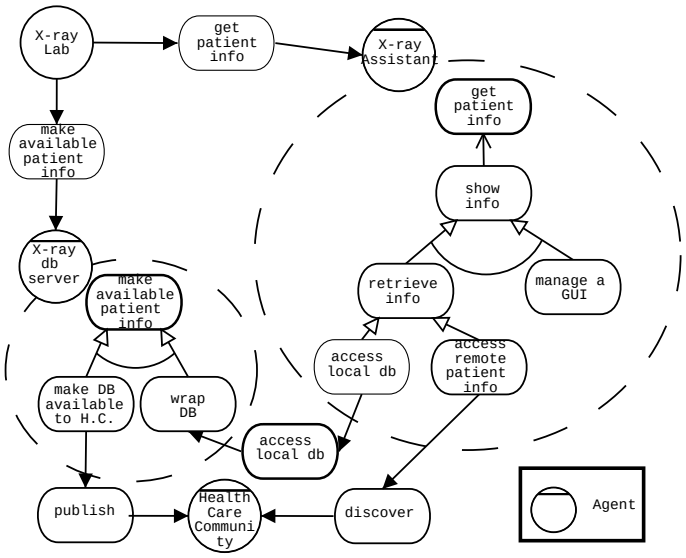


Fig. 11. The X-Ray Lab Peer and the PeerGroup

to-peer infrastructure, JXTA, and multi-agent technologies. We illustrated these guidelines with reference to a scenario taken from the health care domain.

Our long term objective is to complete the framework, working both on the conceptual aspects required to model a DKM application and on the technology, currently being tested in a real-world application.

Acknowledgment

The work presented in the paper is realized inside the *EDAMOK* project, funded by the Provincia Autonoma di Trento (P.A.T.)[6]. We thank John Mylopoulos and Eric Yu for their precious suggestions, and Mark Carman for his review.

References

1. O. Babaoglu, H. Meling, and A. Montresor. Anthill: A framework for the development of agent-based peer-to-peer systems. Technical Report UBLCS-2001-09, University of Bologna, Italy, 2001. http://www.cs.unibo.it/projects/anthill/.
2. M. Bonifacio, P. Bouquet, and P. Traverso. Enabling Distributed Knowledge Management: Managerial and technological implication. *Novatica and Informatik/Informatique*, 3(1), 2002.
3. P. Bouquet, A. Donà, L. Serafini, and S. Zanobini. Contextualized local ontologies specification via ctxml. In *MeaN-02 – AAAI workshop on Meaning Negotiation*, Edmonton, Alberta, Canada, 2002.

[6] `http://sra.itc.it/projects/edamok/index.html`

4. L. K. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos. *Non-Functional Requirements in Software Engineering*. Kluwer Publishing, 2000.
5. F. Giunchiglia, A. Perini, and J. Mylopoulus. The Tropos Software Development Methodology: Processes, Models and Diagrams. In C. Castelfranchi and W.L. Johnson, editors, *Proceedings of the first international joint conference on autonomous agents and multiagent systems (AAMAS02)*, pages 63–74, Bologna, Italy, July 2002. ACM press.
6. F. Giunchiglia, A. Perini, and F. Sannicolò. Knowledge level software engineering. In J.-J.C. Meyer and M. Tambe, editors, *Intelligent Agents VIII*, LNCS 2333, pages 6–20. Springer-Verlag, Seattle, WA, USA, Proceedings of the eighth International Workshop on Agent Theories, Architectures, and Languages (ATAL) edition, August 2001. Also IRST Technical Report 0112-22, Istituto Trentino di Cultura, Trento, Italy.
7. L. Liu, E. Yu, and J. Mylopoulos. Analyzing security requirements as relationships among strategic actors. Technical report, 2002. submitted the IEEE International Symposium on Requirements Engineering.
8. B. Magnini, L. Serafini, and M. Speranza. Linguistic based matching of local ontologies. In *MeaN-02 – AAAI workshop on Meaning Negotiation*, Edmonton, Alberta, Canada, 2002.
9. A. Molani, P. Bresciani, A. Perini, and E. Yu. Intentional analysis for knowledge management. Technical report, ITC-IRST, 2002.
10. Andy Oram, editor. *Peer-to-Peer Harnessing the Power of Disruptive Technologies*. O'Reilly Associates, 2001.
11. S. Patel and A. Sheth. Planning and optimizing semantic information requests using domain modeling and resource characteristics. In *Proc. of the International Conference on Cooperative Information Systems (CoopIS 2001)*, Trento, Italy, September 2001. `http://lsdis.cs.uga.edu/proj/iq/iq_pub.html`.
12. L. Penserini, L. Liu, J. Mylopoulos, M. Panti, and L.Spalazzi. Modeling and evaluating cooperation strategies in p2p agent systems. In *Proceedings of the Agents and Peer-to-Peer Computing, AAMAS02*, 2002.
13. M. Stefanelli. The socio-organizational age of artificial intelligence in medicine. *AI in Medicine Journal*, 23(1):25–48, 2002.
14. P. Szolovits, J. Doyle, and W.J. Long. Guardian angel: Patient-centered health information systems. Technical Report TR-604, MIT/LCS, 1994.
15. M. Wooldridge, P. Ciancarini, and G. Weiss, editors. *Proc. of the 2nd Int. Workshop on Agent-Oriented Software Engineering (AOSE-2001)*, Montreal, CA, May 2001.
16. E. Yu. *Modelling Strategic Relationships for Process Reengineering*. PhD thesis, University of Toronto, Department of Computer Science, 1995.
17. E. Yu and L.M. Cysneiros. Agent-oriented methodologies - towards a challange exemplar. Technical report, Faculty of Information System, University of Toronto, 2002.