The RETSINA MAS, a Case Study

Katia Sycara, Joseph A. Giampapa, Brent Langley, and Massimo Paolucci

The Robotics Institute, Carnegie Mellon University,
5000 Forbes Ave,
Pittsburgh, PA 15213–3890, (U.S.A.)
{katia,garof,blangley,paolucci}+@cs.cmu.edu
http://www.cs.cmu.edu/~{katia,garof,blangley,paolucci,softagents}

Abstract. In this paper we identify challenges that confront the large-scale multi-agent system (LMAS) designer, and claim that these challenges can be successfully addressed by agent-based software engineering (ABSE), which we consider to be distinct from object-oriented software engineering for multi-agent systems (OOSE for MAS) in its consideration of agent *goal*, *role*, *context* and *attitude* as first class objects. We show how we have discovered these principles through our experiences in developing the RETSINA multi-agent system, in implementing specific test applications, and in the derivation of three distinct architectures that help guide and describe the designs of our systems: the *individual agent* architecture, the *functional* architecture, and the *infrastructure* architecture.

1 Introduction

As information technologies become accessible to more and more people and as commercial and government organizations are challenged to scale their services to larger market shares and wider user communities while minimizing or reducing their costs in doing so, there is an increased demand for software applications to provide the following three features to their human end-users:

1. richer application end-to-end functionalities,
2. a reduction of human involvement in "the process" by:
   - reducing information overload,
   - reducing configuration management and system maintenance overheads, and
   - enabling the rapid specification of new, often context-aware tasks,

   and
3. in the combinatorial use of existing software applications and systems in novel or adaptive ways.

Distributed multi-agent system (MAS) technologies and web services show much promise at satisfying the above desiderata by means of their inherent modularity and ease with which they can be recombined to form new applications. When designing new distributed software systems, however, the above broad requirements

and their translations into specific implementations are typically addressed by partial, complementary and overlapping technologies, and the combination of the three gives rise to significant software engineering challenges. Some of the challenges that may arise are: determining the components that the MAS application should contain, organizing the components of the MAS, determining the assumptions that one needs to make in order to implement a MAS application, and if using multiple components off the shelf (COTS), how can their compatibilities with each other, or the degree of effort involved in making them interoperate with each other, be estimated? In this paper we further identify other challenges, and claim that they can be successfully addressed by agent-based software engineering (ABSE). We consider ABSE to be distinct from object-oriented software engineering for multi-agent systems (OOSE for MAS) in its consideration of agent *goal*, *role*, *context* and *attitude* as first class objects. We show how we have discovered these principles through our experiences in developing the RETSINA multi-agent system, in implementing specific test applications, and in the derivation of three distinct architectures that help guide and describe the designs of our systems: the *individual agent* architecture, the *functional* architecture, and the *infrastructure* architecture.

The goal of the design of the RETSINA multi-agent system is to have a positive impact in any combination of the following three areas: (1) to augment a human end user's information-based perceptual capabilities by reducing information overload and providing context-relevant information, (2) to qualitatively and quantitatively improve the range of actions and activities in which the end user can engage, and (3) to enhance the means — typically through the context-aware use of devices, as is done in pervasive and ubiquitous computing — by which humans may perceive the world or by which humans may effect their decisions within it.

The RETSINA MAS is based on the assumption that it will be operating in an open world. The networked environment in which an agent is operating is *open*, or without bounds, it is *dynamic* in nature from the perspective of network topologies, agent capabilities and agent locations, and the networked environment is *uncertain*, that is, the same agent that provided an answer to an earlier request may not be available when called upon again. In the RETSINA MAS there is also an assumption that often there will be some degree of service or functional replication so that should one agent fail, one or many other agents and service providers can be found to substitute for the failed agent. And finally, there is the assumption that many of RETSINA agent behaviors can port to physical robots and produce meaningful results in the physical robotic world.

The RETSINA definition of multi-agent systems is driven by our vision that multi-agent societies should be populated by heterogeneous agents that autonomously organize their own social structures. Our thinking of MAS infrastructure is guided by the desire to enable the flexible design, building and operation of such societies. We consider MAS infrastructure to be the domain independent and reusable substratum on which MAS systems, services, and components live, communicate, interact and interoperate; the infrastructure should

support agents and facilitate their social interactions with each other rather than impose it.

One important consequence is that to achieve heterogeneity the MAS infrastructure should minimize the assumptions that it makes on the agents that populate it, therefore, we make a very strong distinction between the MAS *infrastructure* and the *agent* architecture. We believe that to achieve heterogeneity the MAS infrastructure should not dictate what kind of computational architecture agents have, rather it is up to the agents to find the best way to use the infrastructure to their advantage. Ultimately, the only assumption we make on the architecture of an agent is the awareness of the existence of the MAS infrastructure and of its components, and how to use those components to enable the agent to be part of a multi-agent society, i.e to be socially aware.

The definition of MAS infrastructure that we put forward does not impose any biases with respect to the social structure and coordination regimes of the agents. Indeed, we do not have a "coordinator" component that makes sure that the agents behave socially in a coherent way. Rather we claim that the social structure and coordination regimes should emerge from the behavior of the agents rather than being imposed by the MAS infrastructure. We claim that the MAS infrastructure should be general enough to facilitate any coordination scheme such as team behavior [33, 13], negotiation [17], Contract Nets [27] and auction protocols [8], etc. We feel that a coordination regime as well as social norms [2] are not part of the infrastructure but are particular to the design of a given MAS application society, and are determined by the requirements of the task that the agents are performing. Multi-agent social structure and coordination mechanisms result as applications of the MAS infrastructure rather than being mandated by the infrastructure per se.

The rest of the paper is structured as follows. We begin with a review of the challenges of software engineering for large-scale MASs, and our characterizatoin of agent-based software engineering in Section 2. Section 3 presents a brief description of the three RETSINA architectures. Section 4 discusses some of our lessons learned. We conclude in Section 5.

2 Challenges of SELMAS

The report from the SELMAS workshop [10] lists many challenges, perspectives, and unanswered questions about the nature of large-scale multi-agent systems (LMASs) and agent-based software engineering (ABSE): *how does one define a MAS*, *what are the central issues to be addressed when designing one*, and *is there* really *a difference between ABSE and object-oriented software engineering for MASs?* As strong proponents of ABSE as a unique software engineering paradigm, we would like to briefly discuss what we view as some of the challenges of MAS and LMAS construction, how we characterize ABSE, and what are some of the consequences of these characterizations.

2.1 The Recognized Challenges

Multi-agent systems operate in an open, unbounded world and thus have imposed upon them the requirements that they must be context-aware — aware of how the environment conditions the interactions that the agents can engage in. MAS openness also refers to the actual, implemented agent system, itself. Even if an initial MAS design begins with a limited number of agents, new agents or newer versions of agents will most likely be added to the original MAS at a later date as requirements for the system change, and new features and functions are requested. Because of this openness, there is no single point of resource allocation, synchronization, or failure recovery. The environment is also dynamic and changing, which challenges any software engineering paradigms that require the explicit enumeration of objects and relationships among objects in the environment. The MAS distributed computing environment is uncertain, a characteristic that justifies concerns for partial failure recovery [35]. In distributed computing, where there is no centralized point of control, the failure of any one computation, communication link or network node can render the distributed execution state of the MAS application inconsistent, and the resulting inconsistency may be difficult to identify and thus difficult to remedy. As the number of agents participating in MAS applications increases, the dimensionality of the above concerns becomes combinatorial, and challenges human perception of control and predictability of the system. And, there are different types of heterogeneities that multi-agent systems must address, unpredictably, during their life cycle. Some of the heterogeneities that we have identified and try to accommodate within the RETSINA system are the following:

Communications Heterogeneity Considerations include how many different communications interfaces can or should the agent use for effecting its communications with its peers (e.g. IR, radio, wire, etc.), and what are the available underlying network protocols that will be used.

Coordination Heterogeneity There are multiple coordination techniques [28] such as capability-based coordination [30], team-oriented coordination [14, 33], the Contract Net Protocol [27], auction-based coordination schemes [8, 34], and others, which depend primarily on the task that needs to be performed, and the coordination attitude of an agent (e.g. cooperative, self-interested, antagonistic, etc.).

Environmental Heterogeneity The operating environment can range from the network operating environment, in which considerations focus on how well network protocols are performing (e.g. throughput, transport reliability, network connection permanence, etc.), to the computational environment in which software capabilities change, to physical and terrain environments of agent-augmented hardware and robots.

Functional Heterogeneity This is the identification of agent roles in terms of the services or functions that they contribute to a multi-agent system, and is the focus of the Functional Architecture, described in Section 3.2.

Security Heterogeneity In RETSINA, security is viewed as being parameterized by the application [36], in addition to encryption of communications

and authentication of component identities. Some examples of differences in security models are evaluations of trust per individual agent vs. trust of agents running on a trusted platform, and digital rights management for MAS-aided information fusion, aggregation and sharing.

Semantic Heterogeneity The chief focus of any multi-agent system, this heterogeneity expresses the issue that any two interoperating agents must be certain when using a vocabulary of terms, or translations thereof, that they are using the same concepts with the same relevant inferences of relations as the other communicating agent.

Systems Heterogeneity This heterogeneity is derived from differences of devices and hardware, operating systems, implementation language and execution environment (e.g. within a virtual machine or not), etc., and from the proliferation of versions of all of the above.

Despite these challenges, we believe that practical and non-trivial LMASs can be implemented and executed reliably, by recognizing some important characterizations of ABSE.

2.2 Agent-Based Software Engineering

Our claim is that the engineering principles that motivate a robust and reliable MAS design are those that consider an agent to be defined by its goal, role, context, and attitude. We consider these to be first class objects in the ABSE paradigm, and contend that by not treating these four characteristics as first class objects, object-oriented software engineering founders in its abilities to properly model and predict MAS and LMAS behaviors. We describe these principles as follows.

Goal A *goal* provides the motivation for an agent to perform any activity at all. It constrains the behaviors of individual agents during their interactions with each other, and enables inferencing and predictability of individual agent behavior as well as of the emergent behavior of an entire MAS, itself. We assume that all agents have at least the implicit goals of wanting to announce their existence to the multi-agent community through the advertisements of their capabilities. Another implicit goal that agents may have is the desire to maintain the reliability of their services, which results in them seeking alternative planning and execution strategies should one route to goal completion fail or be perceived to be unlikely.

An agent's goals sometimes might not be completely consistent and place the agent in a dilemma: the agent then must evaluate the tradeoffs involved in pursuing one or another of its goals. For example, an information agent might have the two goals of finding the most current stock information and of acquiring information for free or as cheaply as possible. During its execution, the agent may discover that the free or inexpensive information providing sites respond poorly, or do not provide current information. The agent might need to consider a pay-per-query site that charges for every query, but guarantees a rapid response with the latest data. The agent must evaluate and choose which of the goals it will attempt to achieve.

Role The notion of *role* is confused by the multiple connotations that it has in human languages. We characterize it as a mesh of agent relationships, including authority relationships, that exist within a given context. Authority relationships are important for deriving an agent's range of actions when performing tasks that were delegated to it by a human, and for reasoning about information access and divulgence rights. An agent's role within a given context may change. Roles allow inferences to be made about how an agent will interact with a group of other agents, or about how it will achieve its goals. A characterization of *role* for team coordination, defined in terms of goal, authority, and some contextual indicators, is provided in [14].

Context A *context* establishes the conditions by which an agent's roles, and sometimes goals, can be defined. We do not believe that it is possible to enumerate all the types of contexts in which an agent may be situated, but it is important for an agent to have the ability to recognize when its context has changed. When an agent realizes that it is in a new context, it may need to acquire a new set of roles. Somehow, the agent must be able to determine if the new roles form relationships with roles from other contexts in which the agent belongs, and if so how do the roles from one context impact the decisions and actions of the agent in another context. Consider the example of an agent that provides financial portfolio management advice to an investor, but which also should try to sell certain stock to increase the value of that stock's individual shares. The agent has two distinct roles derived from the two contexts in which it may operate: trusted advisor and interested seller. If the role of the interested seller translates into the context in which the agent is advising, then its role as trusted advisor will be doubted. If the role of seller does not translate to the context of advisor, unless explicitly requested by the client, then the agent as adviser will be trusted by the client.

Attitude The *attitude* of an agent is: its positive or negative disposition to provide reliable and trustworthy services, degree of being deceptive or forthright, the degree to which it will allow its actions to be verified, and its degree of cooperativeness, which can range from altruistic, to self-interested, to being competitive. Most agent systems begin with an implicit assumption about the degree of cooperativeness of their agents.

Not all of the above four characteristics need to be significantly present in a MAS application at the same time. Just as all non-distributed software applications will have varying degrees of functional and dynamic behaviors [22], agents will have varying degrees of: motivation by goal, identity by role, awareness of its context, and dispositions to behave with certain attitudes.

2.3 Consequences

One of the consequences of our characterizations of agent-based software engineering is that agents have grounds by which to make inferences about new priorities and consequences as the open, uncertain, and dynamic agent environment changes. That is, the power of inference enables an agent to be adaptable to

its changing environment. Another consequence of our ABSE characterizations is that one of the desiderata of software engineering, *predictability* of MAS behavior, changes from two perspectives. From the agent perspective, the agent is now *empowered to infer* its own range of actions, perceptions, and expectations for achieving its multiple goals, given: its multiple roles in multiple contexts, its attitudes, and the types of heterogeneities that are in its environment. From the perspective of the MAS observer — which may be another agent, as well — the range of actions and behaviors in which the agent may engage can vary greatly based on the unpredictable contexts, the roles of the agents within those contexts, etc., and therefore ABSE predictability estimates should be performed in terms of ample tolerances rather than in terms of precise specifications with narrow tolerances. Thus, when designing a MAS, an engineer is no longer specifying exactly how an agent will behave, but establishing the bounds and tolerances, or an envelope of acceptable behaviors, by which agents may plan their actions, and by which observers may judge a MAS' behaviors.

The other principle consequences that we will discuss in this paper are the ways in which we translate our ABSE principles into the RETSINA MAS. One of the first derivations of these principles was to adopt a goal-driven hierarchical task network (HTN) deliberative planning system as the general planning architecture for a RETSINA individual agent. By doing so, we could enable agents to plan and replan subtasks and umbrella tasks for achieving their goals. One of the ways in which we tested this scheme was to have a goal-driven, HTN planning agent participate in multiple simultaneous auctions, with different protocols, to achieve the goal of acquiring an object at the "globally" lowest possible price [8].

Another way in which we translate our ABSE principles into reliable implementations is the way in which we use the nature of hierarchical task networks to provide parallel means for composing services, monitoring the execution of distributed computations and fusing the resulting data and control flow as the concurrent computations finish, and automatically determining rollback segments should distributed computations fail partially. We hope that how this happens will be clearer as the reader continues into the next section.

3 The Three RETSINA Architectures

The following three sections provide abstract architectural descriptions that, combined, address many of the heterogeneities listed in Section 2 and that conform to the software engineering principles mentioned in section 2.2. They are the architecture of an individual agent, the functional architecture of a society of RETSINA agents, and the infrastructure architecture that provides the most practical guidance in the systems integration of agent components and technologies.

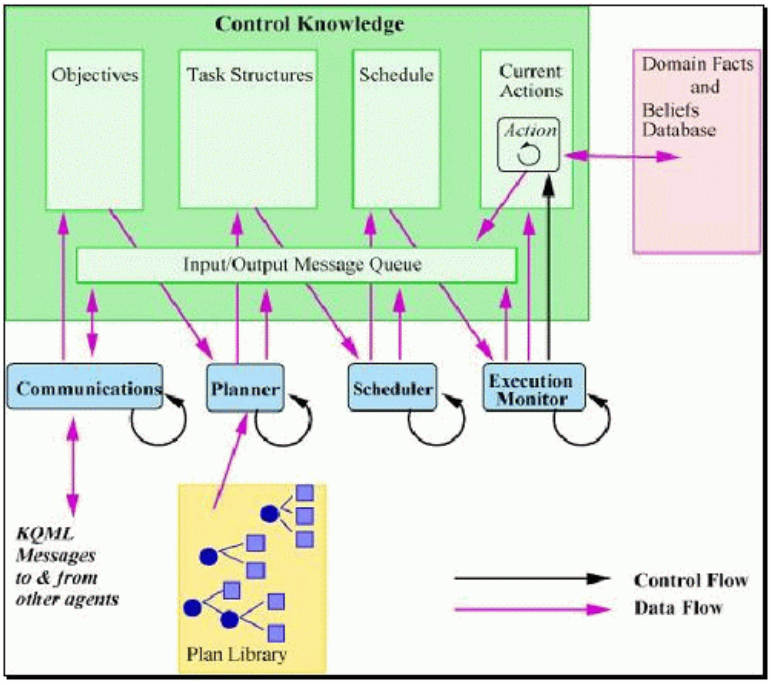

Fig. 1. Schematic diagram of the RETSINA Agent Architecture.

3.1 The RETSINA Individual Agent Architecture

The *RETSINA Individual Agent Architecture* [29, 1, 7] is illustrated by Figure 1. This agent architecture implements hierarchical task network (HTN) planning, scheduling and execution monitoring [21] in three parallel execution threads, while a fourth thread, the *Communicator* [23], provides the means by which the agent communicates with the networked world. The Communicator provides a level of abstraction that insulates the components from issues of agent communication language (ACL), communication session management, the location of agent services, the logging and visualization of agent messages and state information, and the communication transport being used (e.g. infrared, telephone, base band, etc.). The *HTN Planner* thread receives HTN plan *objectives* from the Communicator, extracts the information and instructions contained therein (e.g. an information request, which becomes a current goal of the recipient agent), and attempts to apply the extracted data to all the plans in its plan library. Plan actions are partially *enabled* as the data is applied to them, and once all actions of a plan are completely enabled, they are scheduled by the *Scheduler*. The Scheduler maintains the enabled actions in a priority queue, and works with the *Execution Monitor*, which actually executes the enabled actions, monitors the execution, and handles failures. The coordination among the three planning modules is done in such a way that high-priority actions can interrupt those

being executed by the Execution Monitor, if those being executed are of a lower priority.

3.2 The RETSINA Functional Architecture

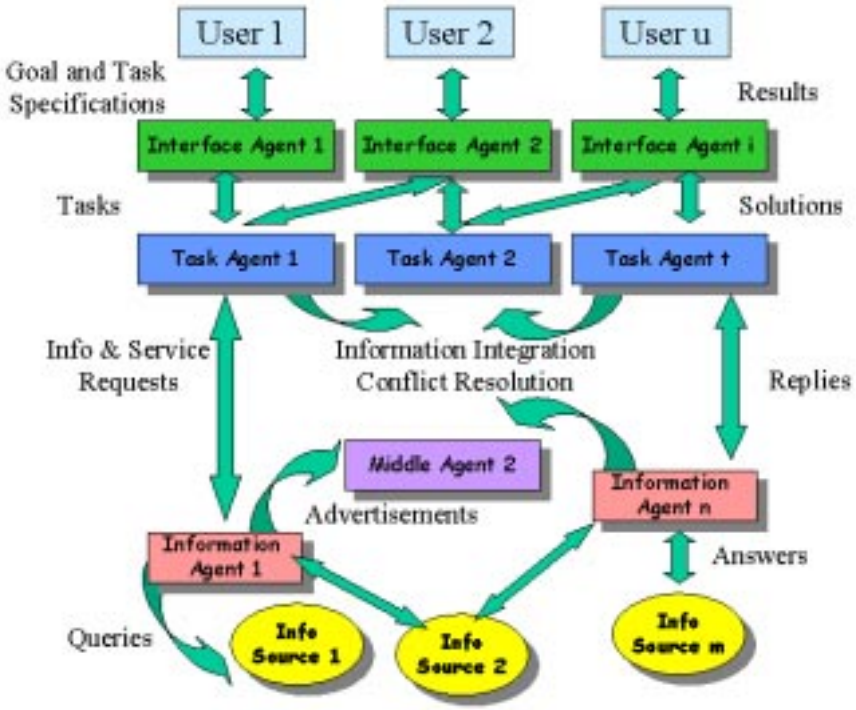

Fig. 2. The RETSINA Functional Architecture

The RETSINA multi-agent system is a collection of heterogeneous software entities that collaborate with each other to provide a result or a service to other software entities or to an end user. Individual agents within that collection have roles which represent their commitment to achieving — or participating in an activity that will achieve — a team goal. Describing agent roles in achieving that goal is easily done by considering the functional contributions that an agent makes, and by focusing on the requester of that contribution (e.g. its anticipated permanence or its frequency of requests) to determine the range of communicative behaviors the service provider agent should support.

The RETSINA Functional Architecture [29] is illustrated by Figure 2, which categorizes agents as belonging to any of four agent types:

Interface agents present agent results to the user, or solicit input from the user. In addition, they could learn from user actions [3]. Interface agents typically represent specific modes of input or output, such as a *VoiceRecognition* agent or a *SpeechGeneration* agent, or can interact with *device agents*[1] to determine the proper way to display information given a device's display characteristics, or to retrieve input from a user. Interface agent behaviors can also be associated with *task agents*.

Task agents encapsulate task-specific knowledge and use that knowledge as the criterion for requesting or performing services for other agents or humans. In this respect, they are the typical agent coordinators of a multi-agent system.

Middle agents [38, 12] provide infrastructure for other agents. A typical instance of a middle agent is the *Matchmaker* [30, 31], or *Yellow Pages* agent. Requesting agents submit a *capability* request to the Matchmaker, which will then locate the appropriate service-providing agents based upon their published *capability descriptions*, known as *advertisements*.

Information agents model the information world to the agent society, and can monitor any data- or event-producing source for user-supplied conditions. Information agents may be *single source* if they only model one information source, or may be *multi-source* if one information agent represents multiple information sources. Information agents can also update external data stores, such as databases, if appropriate.

By classifying agents functionally, we believe that it is possible to uniformly define agent behaviors [7] that are consistent with their functional description. For example, information agents implement four behaviors for interacting with the data sources that they model: *ask once*, *monitor actively*, *monitor passively*, and *update*. RETSINA agents typically use the capability-based coordination [28] technique to task each other, which means that one agent will dynamically discover and interact with other agents based on their need and based on the other agents' capability descriptions. RETSINA agents also support other forms of coordination techniques, such as team-oriented coordination [14], auction-based coordination [8], and Contract Net Protocol.

3.3 The RETSINA Infrastructure Architecture

Agents in a MAS are expected to coordinate by exchanging services and information, to be able to follow complex negotiation protocols, to construct models of each other and shared models of their tasks and world, to agree on commitments, and to perform other socially complex operations. In order to interact robustly, agents need an infrastructure of services that enable them to, for example, find each other in open, ever changing and uncertain environments, to communicate, and to warrant that the proper security constraints are satisfied,

[1] Device agents are a type of information agent that represent systems information about devices to a MAS. Some RETSINA device agents that have been written are for PDAs (e.g. PalmPilots and iPAQs), and WAP-enabled cell phones.

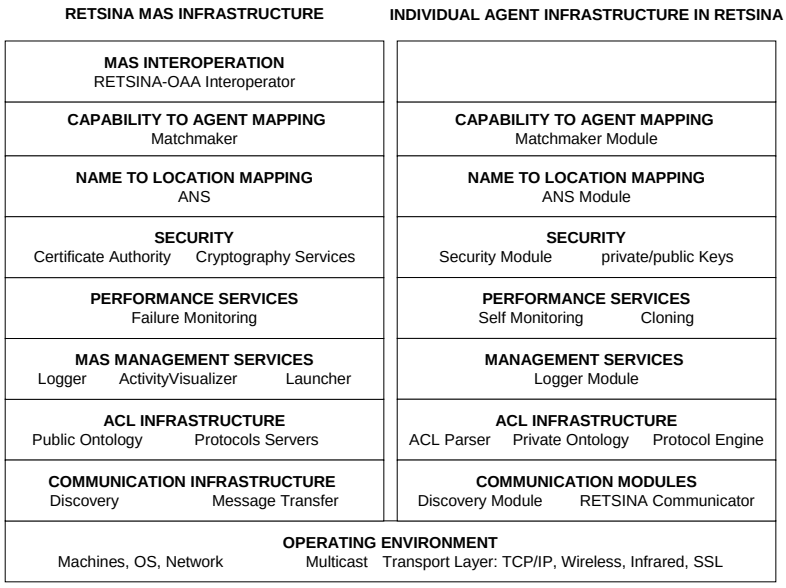

Fig. 3. The RETSINA MAS Infrastructure and Individual Agent Infrastructure

etc. In addition, agents need conventions, such as Agent Communication Languages (ACLs), conversational policies and ontologies that define the meaning of the terms agents use to provide the basis for achieving semantic interoperability and agreement with each other. Moreover, agents need to share knowledge of how to use the infrastructure, ACLs, and protocols. In this section we analyze some of the requirements of the MAS infrastructure as it emerged from our experiences with the RETSINA MAS. The reader is referred to [32] for a more detailed explanation.

Figure 3, which describes the organizational architecture of RETSINA, represents the abstract dependencies of the upper layers of infrastructure modules on the lower layers, for each of the two columns[2]. The figure is organized to show, in the left column, the organization of the MAS , and in the right column, how the overall architecture of the MAS is reflected in the internal modules of an individual agent so that it may interact with the infrastructure components. While the diagram shows the settings adopted by the RETSINA MAS, we believe that every MAS implements, in whole or in part, implicitly or explicitly,

[2] For the sake of clarity, this diagram is a simplification of our own intuitions and some details are not accurately represented. For example, the placement of the security level should either span or be a component of multiple layers to imply that security is a feature of the whole MAS. Similarly, *performance services* are also a feature of *capability to agent mapping*.

the modules described by the diagram, thus our belief that every MAS can ultimately be mapped to this diagram. Note that we do not impose the requirement or expectation that every agent must have modules that support each of the infrastructure components: an agent implementation may include some modules and not others. Nor does the infrastructure dictate how these modules are implemented or how they should interact with each other. Rather, as has been our experiences, the infrastructure serves as a reference to guide in understanding and placing different information technologies in an architecturally sound manner, and in informing the ways in which interoperability may be achieved between systems of differing MAS architectures [12]. In the paragraphs that follow we briefly describe the layers of the left column, from the bottom, up.

Operating Environment. The operating environment may force a decision about programming language, communication protocols, network bandwidth and persistence. RETSINA has supported agents running under a wide variety of conditions, such as: different operating systems and versions thereof, spanning from Windows and WindowsCE, to Linux, to Sun OS, to Palm OS, to WAP-enabled cell phones; agents implemented in many different languages including: Java, C and C++, and Lisp, to mention a few; and agents communicating directly in: TCP/IP, UDP, HTTP, wireless, SSL, infrared, across mobile sockets [18], low-power radio like BlueTooth[3], and the X-10[4] powerline carrier protocol.

Communication Infrastructure. The communication infrastructure layer defines an abstract communication channel for intra-agent information exchange. It should be medium-independent to allow agents to communicate on any physical transmission medium, and ACL independent to be parametrized to function under any condition. The RETSINA Communicator [23] makes use of different network protocols such as TCP/IP and multicast; in addition, through the DARPA Grid, RETSINA makes use of the Jini[5] communication infrastructure based on remote method invocation.

ACL Infrastructure. The ACL infrastructure layer specifies the language spoken by the agents in the MAS, for example KQML [9] or FIPA[6], as well as conversational policies [15, 26] agents should adhere to in their interactions. Furthermore, the ACL infrastructure should specify the semantics associated with the language, whether it is based on a mentalistic model [16] or on a social interaction model [25]. The RETSINA MAS uses a subset of KQML as the typical ACL for all the agents in the MAS, but the modular design of the RETSINA Communicator allows agent programmers to easily use other ACLs, such as FIPA, or novel languages that are based on XML [8, 4, 5].

Management Services. Management services can support the analysis of agent activity to support optimization or debugging in the case of failures, and since the deployment of a MAS requires the deployment of a number of agents, they also should support launching capabilities to minimize the burden of de-

[3] `http://www.bluetooth.com/`
[4] `http://www.x10.org/`
[5] `http://www.sun.com/jini`
[6] Foundation for Intelligent Physical Agents, `http://www.fipa.org/`.

ploying a MAS and to make the management of large MASs scalable for humans. Within the RETSINA MAS infrastructure a number of management tools have been developed to support activity monitoring, visualization and launching [11]. It should also be noted that the infrastructure that supports agent location and discovery also addresses a significant portion of the activities involved in MAS configuration management.

Performance Measurement. The performance measurement layer provides a set of protocols and tools that allow an agent to learn about the trust and reliability of other agents. Agents should also monitor their own behavior to verify that they are meeting their expected requirements. Self monitoring capabilities have led to the implementation within RETSINA of agents that are able to clone themselves [24] when overwhelmed by many tasks delegated to them to increase the likelihood that they can fulfill all their assigned tasks.

Security. We identified two dimensions of security: communication security and infrastructure integrity. Communication security guarantees privacy so that a message cannot be eavesdropped, authentication so that an agent is certain of its partner's identity, and non-repudiation, which prevents agents from denying that they took part in a transaction. Infrastructure integrity guarantees that no agent can manipulate the information stored in the infrastructure components such as the Agent Name Server or the Matchmaker, and that the contents of a message cannot be changed by an unauthorized agent. RETSINA supports communication security through the use of a Certificate Authority that provides unique public and private keys to agents in the MAS [37]. Infrastructure integrity, has been achieved by strictly controlling the operations an agent can do against the infrastructure components to protect their load and prevent unexpected uses. It has also been studied formally in the way in which it applies to protocols for electronic commerce [36].

Name to Location Mapping and **Capability to Agent Mapping.** The *Name to Location Mapping* module maps an agent identifier (e.g. name) to the specific location where an agent may be found. The *Capability to Agent Mapping* refers to processes of semantic matchmaking [30, 31], which were described in Section 3.2. Although, for reasons of space, we cannot elaborate on the many complexities and heterogeneities that are embodied by these layers, these two layers are at the heart of a robust and scalable MAS.

MAS Interoperation. This modules permits multiple agents of different MAS architectures to interoperate with each other despite their architectural differences. As mentioned earlier, interoperability across MAS boundaries is based on the identification and exploitation of similarities between the two agent architectures, across multiple infrastructure levels. Details of the issues involved are provided in [12]. In addition to interoperating with the Open Agent Architecture (OAA) [20] and DARPA Grid [6] MASs, we have also developed interoperators with Jini services.

4 Lessons Learned

Given the breadth and scope of our experiences in developing the RETSINA system and reference architectures, there are many lessons learned, evaluations of our architectures, and metrics that we could discuss. Given limitations of space, we limit our lessons to how we deal with heterogeneity, and how we respond to the three questions which motivated the workshop.

4.1 Dealing with Heterogeneity

One of the lessons learned from designing the RETSINA MAS for the types of heterogeneity listed in Section 2.1 is the confirmation that there are practical merits to designing a MAS for such levels of heterogeneity. The practical merits are: the ease by which components can be combined to achieve better scalability, the achievement of new functionality, and the achievement of novel functional dimensions. The following are some example derivations of MAS functionality that were accommodated by the design of the RETSINA MAS. That is, these software artifacts, listed according to the heterogeneity that they address, were implemented *without* needing to retroactively modify any of the existing RETSINA architecture.

Communications Heterogeneity Separating the implications of contractual commitment in agent-to-agent requests made it possible to interoperate between any RETSINA and OAA agent when designing the $RETSINA - OAA\_InterOperator$ [12]. In contrast, the lack of such separation inhibited scalability of agent interoperations between two other agent systems in the same experiments. Also, by using a "loosely coupled" agent message passing scheme as opposed to a "tightly coupled" remote method invocation, RETSINA agents need only "pick and choose" the arguments and parameters that they need or can translate, and can ignore the rest if they are not essential to the computation. Nor is it necessary to modify or have access to an agent's program code to effect new communications between two previously existing agents.

Coordination Heterogeneity In RETSINA, the coordination model is provided by each individual task agent[7], which employs the most appropriate coordination model for its task. Since task agents can be added and executed as needed, RETSINA allows for a wide variety of agent coordination models to interoperate with each other.

Environmental Heterogeneity We have describe schemes by which agents that represent physical robot capabilities help determine role assignments of the robots in a variety of heterogeneous team-coordination tasks [13, 14]. Our location discovery infrastructure, e.g. local and hierarchical agent name service, local and wide area discovery, and viral agent-to-agent community formation services [19] provide agents with a variety of ways to find each other in different network topologies and administrative domains.

[7] See section 3.2.

Functional Heterogeneity A simple extension of device agents to include specific input and output properties of their host device enabled RETSINA interface agents to search for the most appropriate display device for a user in a computer rich environment. Alternatively, if the user has limited display capabilities available, such as limited screen real estate on a WAP-enabled cell phone or PDA, then the RETSINA interface agents would use that device agent-provided information to modify the display of information[8].

Semantic Heterogeneity In addition to the matchmaking approach to achieving semantic interoperability among agents and services [5, 30, 31], there are also issues of different implementation-dependent representational schemes, such as those encountered in the translation of RETSINA and OAA advertisements [12].

Systems Heterogeneity To assist systems administrators in the allocation, configuration, launching and monitoring of MAS applications in a heterogeneous computing environment, we developed RECoMa, the RETSINA Configuration Manager [11]. RECoMa uses the RETSINA infrastructure discovery services, capability-based matchmaker, and device agents that represent the capabilities and resource loads of the heterogeneous computing platforms on which the agents are running, in order to help systems administrators perform their duties.

4.2 Lessons Relevant to the Three Workshop Goals

The report from the SELMAS workshop [10] identified three concerns as the goals of the workshop: (1) to determine the overlap and the integration of agent-based software engineering (ABSE) and object-oriented software engineering for multi-agent systems (OOSE for MAS); (2) to understand the issues in the agent technology that hinder or improve the production of large-scale distributed systems; and (3) to provide a comprehensive overview of software engineering techniques that may successfully be applied to deal with the complexity associated with realistic multi-agent software.

To address the first concern, our perspective is that ABSE focuses MAS design consideration on *what* declaratively defines and motivates the agent system, whereas OOSE for MAS focuses on *how* those motivations will be achieved and implemented. From the RETSINA perspective, the principles of goal, role, context and attitude are motivated by the task, which is represented in the MAS by the task agents. Task agents provide the overall coordination for any ad hoc assembly of agents, so it is fitting that the principles of agent-based software engineering motivate considerations that define their nature. For example, consider how all four first class ABSE principles simultaneously enter into consideration of whether a task agent will need to cooperate as a peer with other task agents and therefore need to define and understand notions of subgoals and roles for itself and peers, while operating in an agent social context of peer-to-peer cooperation with a presumed attitude of mutual trust and desire to collaborate. Such

[8] `http://www.cs.cmu.edu/ softagents/mocha.html`

considerations will motivate the need to employ mechanisms of team-oriented plan negotiation, revision and execution monitoring. Contrast these considerations, for example, with those that a task agent might have if it is participating in a competitive auction in which its bidding might be negatively influenced by devious agents.

We perceive OOSE for MAS, on the other hand, as informing the interactions of the components of the individual layers of the RETSINA Infrastructure Architecture, in which the issues of reliability and predictability in the traditional OOSE sense apply to the interactions between the individual agent and the MAS infrastructure as a whole, and vice versa. By mapping OOSE for MAS to this architecture, we feel that we respond directly to the concerns of how to model MAS, what design standard methodologies are appropriate for MAS without misusing the abstractions that they provide, and to the overall question of how ABSE and OOSE for MAS differ.[9]

In response to the second concern, it has been our experience that the issues which hinder agent technology are derived from the many heterogeneities, complexities and views that confront the designer of a large-scale MAS, particularly for open, uncertain and dynamic computing environments. The first step, in a way, is to know when to focus on them, and when to know that many of these heterogeneities and complexities will be addressed automatically by MAS infrastructure. We feel that MAS designers can derive inspiration for identifying their heterogeneities and for estimating how they will be resolved by our discussions and examples in sections 2.1 and 4.1.

MAS designers can benefit from many off-the-shelf technologies, but knowing how to map those technologies to a MAS implementation sometimes imposes significant overheads to MAS designers. Hence, we view MAS infrastructure and agent libraries to interact with that infrastructure, as is present in the RETSINA libraries, and MAS architecture maps such as the RETSINA Infrastructure Architecture, as critical enablers to the scalability of large-scale distributed systems.

Regarding the tacit question of what makes a MAS *large-scale*, we take the position that issues of scale in terms of number of stake holders and numbers of agents need to be evaluated in light of the tasks that motivate the agents' interactions. Based on parallels with human social institutions, tasks are iteratively decomposed into subtasks and large populations into smaller communities until the proper dimensions of task complexity and participants are reached. In other words, issues of interaction complexity of large numbers of agents in complex tasks are often reduced by the human approach to solving the task, so it is difficult to study in the abstract. Those scalability issues that can be studied in absence of specific tasks are those that relate to infrastructure scalability, which often provides services of search, discovery, location, and community formation. And, given mechanisms of formal analysis and protocol simulation systems, these types of analyses are feasible without actually implementing large-scale MASs.

[9] See the bullet, *How is multi-agent software engineering different from object-oriented software engineering*, in section **8. Workshop Discussions, Lessons Learned and Lines for Future Research** of the workshop report [10].

While the papers of the workshop respond collectively to the third workshop goal, we would like to add an observation that was not reflected in the workshop summary but does respond to both the second and the third goals. Namely, by "empowering" agents with the autonomy to pick and choose with which other agents and services they will work, distributed network application designers are "delegating" significant portions of their design and execution overheads to automation. When this happens, the emphasis of design considerations shift appropriately to issues of capability representation, recognition, and understanding, and to the specifications of tolerances by which agents and services will be composed and evaluated. In a few of our test application scenarios we have seen how the enabling of agents to dynamically locate each other across a variety of network topologies and administrative domains significantly reduces the human overhead of managing the execution of non-trivial MASs in a partially redundant, heterogeneous computing environment [11].

5 Conclusions

The contributions of this paper are multiple, from multiple perspectives. First, we identified some high-level challenges that face multi-agent system designers, and identified seven types of heterogeneities that render the challenges more concrete in addition to providing some initial metrics by which MAS architectures can be evaluated. Second, in our belief that agent-based software engineering is distinct from object-oriented software engineering for MASs, we named and described the four principles that should be considered as first class objects in ABSE: goal, role, context and attitude. A key motivation for identifying these objects as *first class* is the generality with which they can determine which MAS features will be needed for a specific MAS application. Third, we were able to describe the three RETSINA architectures, combined, in a way that illustrated their mutually complementary nature, and which also provided an indirect validation and verification of our choice of heterogeneity types. And a fourth contribution is the recognition that through our descriptions of challenges, heterogeneities, and the RETSINA architectures, we were able to respond to the three goals of the workshop, and to observe how MAS infrastructure and design principles also reduce the dimensionality of ABSE preoccupations for MAS designers. Although ABSE is in its infancy and is therefore relatively less articulate of its precepts than traditional and object-oriented software engineering, it certainly has features that make it a unique discipline in its own right, and promising in the guidance that it can offer to the reliability of large-scale multi-agent systems.

6 Acknowledgments

The authors would also like to acknowledge the contributions of past and present members[10] of the Intelligent Software Agents Laboratory at CMU, who provide

[10] `http://www.cs.cmu.edu/~softagents/people.html`

the many ideas and enthusiasm that make this research possible. This research was sponsored in part by the Office of Naval Research Grant N-00014-96-16-1-1222, by DARPA Grants F30602-98-2-0138 and F30602-00-2-0592, by the Air Force Office of Scientific Research Grant F49620-01-1-0542, by the National Science Foundation Award IIS-0205526, and by National Aeronautics and Space Administration Grant NCC21317.

References

1. D. Brugali and K. Sycara. Agent technology: A new frontier for the development of application frameworks? In M. Fayad, D. Schmidt, and R. Johnson, editors, *Object-Oriented Application Frameworks*. John Wiley, 1998.
2. C. Castelfranchi. Modelling social action for AI agents. *Applied Artificial Intelligence*, 103:157–182, 1998.
3. L. Chen and K. Sycara. WebMate: A personal agent for browsing and searching. In *Proceedings of the Second International Conference on Autonomous Agents and Multi-Agent Systems (ICMAS-98)*, May 1998.
4. DAML-S Coalition. DAML-S: Semantic markup for web service. In *Proceedings of the International Semantic Web Workshop (SWWS-01)*, 2001.
5. DAML-S Coalition. DAML-S: Web service description for the semantic web. In *The First International Semantic Web Conference (ISWC-02)*, 2002.
6. DARPA CoABS Program. Grid web site. http://coabs.globalinfotek.com/, 2000.
7. K. Decker, A. Pannu, K. Sycara, and M. Williamson. Designing behaviors for information agents. In *Proceedings of the First International Conference on Autonomous Agents (Agents 1997)*, February 1997. 0–89791–877–0/97/02.
8. G. Economou, M. Paolucci, M. Tsvetovat, and K. Sycara. Interaction without commitments: An initial approach. In *Proceedings of the Fifth International Conference on Autonomous Agents (Agents 2001)*, 2001.
9. T. Finin, Y. Labrou, and J. Mayfield. KQML as an agent communication language. In J. Bradshaw, editor, *Software Agents*. MIT Press, Cambridge, 1997.
10. A. F. Garcia and C. J. P. de Lucena. Software engineering for large-scale multi-agent systems SELMAS 2002. *ACM Software Engineering Notes*, 27(5):82–88, September 2002.
11. J. A. Giampapa, O. Juarez-Espinosa, and K. Sycara. Configuration management for multi-agent systems. In *Proceedings of the Fifth International Conference on Autonomous Agents (Agents 2001)*, pages 230–231. Association for Computing Machinery, June 2001. ISBN: 1-58113-326-X.
12. J. A. Giampapa, M. Paolucci, and K. Sycara. Agent interoperation across multagent system boundaries. In *Proceedings of the Fourth International Conference on Autonomous Agents (Agents 2000)*. Association for Computing Machinery, June 2000. ISBN: 1-58113-230-1.
13. J. A. Giampapa and K. Sycara. Conversational case-based planning for agent team coordination. In *Case-Based Reasoning Research and Development: Proceedings of the Fourth International Conference on Case-Based Reasoning (ICCBR 2001)*, volume 2080, pages 189–203, Berlin Heidelberg, July 2001. Springer-Verlag.
14. J. A. Giampapa and K. Sycara. Team-oriented agent coordination in the RETSINA multi-agent system. Technical Report CMU-RI-TR-02-34, The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, December 2002. Presented at AAMAS 2002 Workshop on Teamwork and Coalition Formation.

15. M. Greaves, H. Holback, and J. Bradshaw. What is a conversation policy? In *Agents-99: Workshop on Specifying and Implementing Conversation Policies*, 1999.
16. M. J. Huber, S. Kumar, P. R. Cohen, and D. R. McGee. A formal semantics for proxy communicative acts. In *Agent Theories, Architectures and Languages (ATAL-01)*, 2001.
17. N. Jennings, K. Sycara, and M. Wooldridge. A roadmap of agent research and development. *Journal of Autonomous Agents and Multi-Agent Systems (JAAMAS)*, 1(1):275–306, 1998.
18. B. Kuntz and K. Rajan. MIGSOCK: Migratable TCP socket in Linux. Master's thesis, Information Networking Institute, Carnegie Mellon University, February 2002. TR 2001-4.
19. B. Langley, M. Paolucci, and K. Sycara. Discovery of infrastructure in multi-agent systems. In *Agents-2001 Workshop on Infrastructure of Agents, MAS and Scalable MAS*, 2001.
20. D. Martin, A. Cheyer, and D. Moran. The open agent architecture: A framework for building distributed software systems. *Applied Artificial Intelligence*, 13(1–2):92–128, 1999.
21. M. Paolucci, O. Shehory, and K. Sycara. Interleaving planning and execution in a multiagent team planning environment. Technical Report CMU-RI-TR-00-01, The Robotics Institute, Carnegie Mellon University, 2000.
22. J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modeling and Design*. Prentice-Hall International, Inc., 1991.
23. O. Shehory and K. Sycara. The RETSINA Communicator. In *Proceedings of the Fourth International Conference on Autonomous Agents (Agents 2000)*, 2000.
24. O. Shehory, K. Sycara, P. Chalasani, and S. Jha. Increasing resource utilization and task performance by agent cloning. In M. S. V. A. Rao and M. Wooldridge, editors, *In Lecture Notes in AI: Intelligent Agents*. Springer Verlag, 1998.
25. M. P. Singh. Agent communication languages: Rethinking the principles. *IEEE-Computer*, 11, 1998.
26. I. Smith, P. Cohen, J. Bradshaw, M. Greaves, and H. Holmback. Designing conversation policies using joint intention theory. In *Proceedings of the Second International Conference on Autonomous Agents and Multi-Agent Systems (ICMAS-98)*. IEEE Press, 1998.
27. R. G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, 29(12):1104–1113, 1980.
28. K. Sycara. Multiagent systems. *AI Magazine*, 19(2):79–92, Summer 1998.
29. K. Sycara, K. Decker, A. Pannu, M. Williamson, and D. Zeng. Distributed intelligent agents. *IEEE Expert, Intelligent Systems and their Applications*, 11(6):36–45, 1996.
30. K. Sycara, K. Decker, and M. Williamson. Middle-agents for the internet. In *IJCAI-97*, 1997.
31. K. Sycara, M. Klusch, S. Widoff, and J. Lu. Dynamic service matchmaking among agents in open information environments. *Journal ACM SIGMOD Record, A. Ouksel, A. Sheth (Eds.)*, 28(1):47–53, March 1999.
32. K. Sycara, M. Paolucci, M. van Velsen, and J. Giampapa. The RETSINA MAS Infrastructure. *Joint Special Issue of Autonomous Agents and MAS*, 7(1–2), July 2003. forthcoming.
33. M. Tambe. Towards flexible teamwork. *JAIR*, 7:83–124, 1997.

34. M. Tsvetovat, K. Sycara, Y. Chen, and J. Ying. Customer coalitions in the electronic marketplace. In *Proceedings of the Fourth International Conference on Autonomous Agents (Agents 2000)*, June 2000.
35. J. Waldo, G. Wyant, A. Wollrath, and S. Kendall. A note on distributed computing. Technical Report SMLI TR-94-29, Sun Microsystems Laboratories, 1999.
36. H.-C. Wong. *Protecting Individuals' Interests in Electronic Commerce Protocols.* PhD thesis, Computer Science Deptartment, Carnegie Mellon University, 2000. CMU-CS-00-160.
37. H. C. Wong and K. Sycara. Adding security and trust to multi-agent systems. In *Agents-99 Workshop on Deception, Fraud and Trust in Agent Societies*, 1999.
38. H.-C. Wong and K. Sycara. A Taxonomy of Middle-agents for the Internet. In *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems (ICMAS-00)*, 2000.