Using Multi-Agent Based Middleware to Implement a Distributed Peer-To-Peer Semantic Service Oriented Architecture

Daniel Bertinshaw

Department of Computer Science, University of Auckland,
Private Bag 92019, Auckland 1142, New Zealand
dber021@aucklanduni.ac.nz

Abstract. In this work we give the outline for a novel approach to semantic web service based applications. This approach centers around the use of so-called intelligent agents as the basis for the middleware layer. We give a broad outline of our prototype architecture and preliminary correctness results.

Keywords: Semantic Web Services, Service Oriented Architecture, Multi-Agent System, Middleware, Service Composition

1 Motivation

The Semantic Web and Semantic Service Oriented Architectures[6, 4] promise to create systems that can re-use components by composing the resources available on the semantic web. The semantic web resources being pages of data or services that process data.

This particular idea of composition of services provides an advancement over the traditional notion of service oriented architectures: the services available can be loosely coupled and automatically re-used in a dynamic fashion to achieve ad-hoc tasks. These properties of re-used and ad-hoc automation in turn promise to reduce workload on engineers, allowing the instrumentation of common and simple tasks from a basic definition of the task itself.

The key to the possibility of the automated task composition is that the services of the semantic web differ from their non-semantic cousins by having documents in a machine readable format that describe functions the service performs. This document allows automatic reasoning, planning and workflow execution systems to inspect and decide if a service can be included in a dynamic fashion.

The majority of these systems in place (such as WSMX and UDDI-based systems) use some form of central repository to handle the publication and subscription of a service, while keeping the integration (grounding) of clients and providers distributed. This approach however has problems with maintaining the scale of the central service registry as the number of available services increases.

A further problem we have identified with this approach is managing the interactions of consumers and providers: integration happens in a tightly coupled fashion outside of good design patterns.

2 Goals

The goal of our research is to build a system that allows dynamic composition of services for ac-hoc tasks to be achieved in a fully distributed fashion, removing the need for a central repository to find and compose the patterns. Additionally we wish this system to be able to manage and broker interactions between services within a distributed environment. This allows increasingly complex tasks to utilise more physical computing resources of the system in an ad-hoc manner. With these aims in mind we define four main goals to this research:

- 1) that there is no central system for service registration or lookup
- 2) that the planning and execution of a workflow of services can be distributed across the available physical resources of the system
- 3) that interaction between services and other resources is loosely coupled.
- 4) that the system is able to track and manage tasks being executed across itself.

We anticipate that this will allow semantic service oriented architectures, such as semantic grids, to efficiently carry out ad-hoc tasks while allowing the timely inclusion of new services and data into tasks as they are assigned to the system.

3 Approach

We have chosen to investigate the use of multi-agent systems as a form of middleware to enable the distribution of system capabilities and achieve the four goals of this research as outlined previously. Multi-Agent Systems have been used with semantic web applications in the past[3], but we propose to employ them in the middleware and integration layers in a Semantic Service Oriented Architecture (SSOA).

To take SSOA and incorporate agents into it, we propose the construction of a novel architecture using a multi agent system as an intelligent middleware layer that brokers the services based on their semantic descriptions and manages the interactions between them.

In our architecture we have services, which are compiled code in the form of annotated classes held locally or integration with remote services (with semantic descriptions of the service being held by the agent in a local service registry). This means every agent has a registry of its own services and knowledge that it maintains. Should another agent require some capabilities it broadcasts this requirement to the set of available agents. Those agents that have the ability to fulfill the requirement propose solutions to the requesting agent, as shown in

Fig. 1. This allows services to be represented by their agent as a snapshot of its capabilities. If a service's capability changes, so does the agent's entry for that service which makes the system tolerant to changing services.

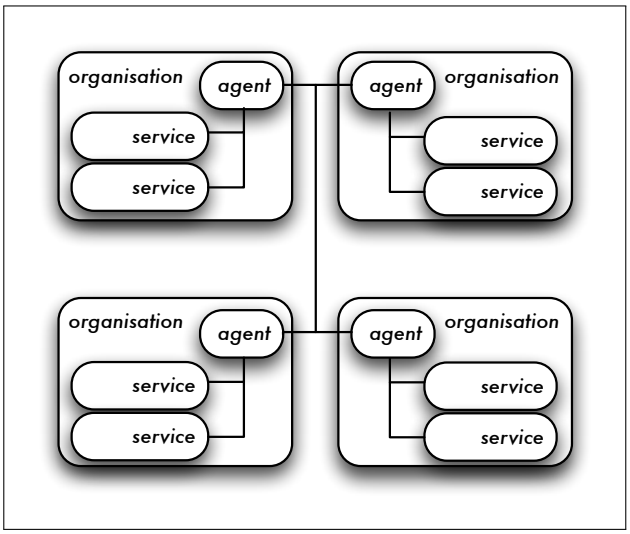

Fig. 1. Our proposed Service Oriented Architecture, with multi-agent based middleware layer

The management of interactions is done by sending a task requirement to an agent along with the data that the task is to be performed on, or a description of the data to be found. The agent then uses this description to automatically plan a workflow incorporating discovery of relevant data and the use of relevant services. By having the services and users communicate intentions and requirements to the agents, the agent can utilise available resources according to a decision making process. In this way the instructions to achieve a task need not be hard coded, which further allows the system to be loosely coupled.

To provide a loose coupling mechanism for the services that the agents broker for we propose an API based on Java annotations that can be used to add First Order Predicate Logic to the methods and data presented in a Java class. Using annotations embeds the semantics within the source code, removing the requirement of an external document for the semantics of the class. These annotations are then transformed (by introspection using reflection at load time for the class) into descriptions stored in a local repository that can be used by the agent's internal reasoner to add to task compositions if they match a required sub-task.

The annotations of the services, as well as facts known about the world, are used by the reasoning system of the agent to establish if a goal can be satisfied. The ontology is a set of documents that express facts about the world, but is

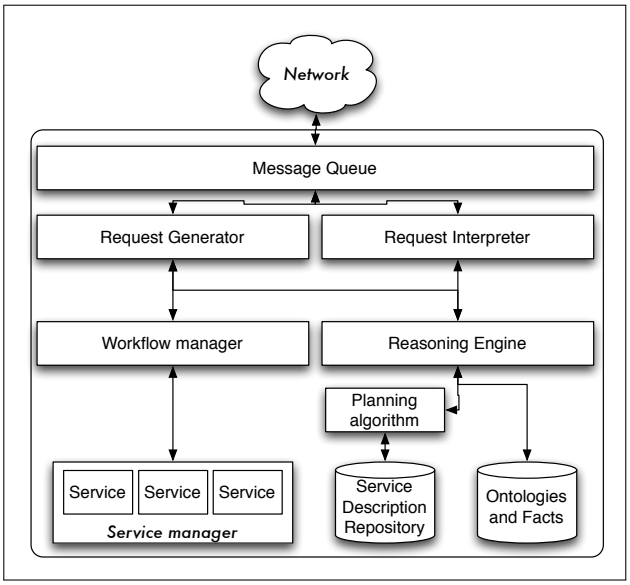

Fig. 2. Internal architecture of an agent

not a complete representation. Each agent has its own domain that it operates in. If a goal cannot be achieved because an agent lacks complete knowledge, a request to solve that problem is sent to the other agents of the collective. This way, even if an agent does not have complete knowledge, the system does and can use knowledge to reason if any agent has this knowledge. The reasoning engine is responsible for checking that tasks can be understood by an agent and to specify the plan requirement to the planning algorithm. This structure is shown in Figure 2. This ensures that our system has perfect knowledge, which is assured in centralised systems.

Using a multi-agent system to implement the middleware allows us to use several existing solutions to issues that occur with the planning and execution of ad-hoc tasks, and especially those that are intrinsic to distributed systems. These include solutions to planning[1], communication[2], collaboration and execution of tasks[5].

As multi-agent systems can be used to distribute the load of planning algorithms, our approach meets goals 1 and 2: there is no central system, and the agents can collaborate to plan and execute solutions to tasks. Since services are invoked (based on descriptions) by the agent and never directly based on a reference, this is also a loosely coupled system, satisfying goal 3. Lastly, agents can also manage which services are executed, when and how often, according to a set policy, which achieves goal 4.

This architecture scales to match the number of organisations in using the system: as each organisation has an agent, the number of agents available increases proportionally. This does not take into account load on any single agent

though. Each agent is not an atomic construction, they are a single logical endpoint for data and single repository for data but are not constructed as a single program on a single machine. In this way our architecture allows single agents to adapt to heavy data usage.

4 Conclusions and Future Work

We have given a novel architecture that places the "intelligent" component of a semantic service oriented architecture (the elements that performs the reasoning and planning) into a middleware layer based on multi-agent systems. We anticipate this will achieve the four goals we stated for an SSOA by reducing the problem to a multi-agent system, which are known to be able to achieve the four stated goals.

We have made initial mathematical verifications of this architecture and have begun the construction of a prototype based on a robust set of open source technologies. This prototype has yet to be tested for performance and consistency in controlled experiments against systems that perform the same job of ad-hoc task composition.

In future we plan to deliver a demonstrable prototype, based on an existing real world domain. This prototype will be used for demonstration, testing and tuning purposes.

We plan to evaluate the agent-service integration component for performance and consistency by testing the performance of the agent as services and requests increase. Additionally we plan to test the agent collaboration by simulating a variable number of organisations solving tasks for a real world domain.

We aim to use our results to show the real world usefulness of this system.

References

1. Chen, Z.: Distributed collaborative production planning: Multi-agent system framework and coordination technology. In: Information Management, Innovation Management and Industrial Engineering, 2009 International Conference on. vol. 3, pp. 619 –624 (26-27 2009)
2. Duan, L., Zeng, G., Huang, L., Feng, Y.: Xml-based multi-agent communication system architecture. In: Genetic and Evolutionary Computing, 2008. WGEC '08. Second International Conference on. pp. 227 –231 (25-26 2008)
3. Hendler, J.: Agents and the semantic web. IEEE Intelligent Systems 16(2), 30 – 37 (2001)
4. Hendler, J., Lee, T.B., Miller, E.: Integrating applications on the semantic web. Journal of the Institute of Electrical Engineers of Japan 122(10), 676 – 680 (2002)
5. Marc, F., Degirmenciyan-Cartault, I.: Multi-agent planning as a coordination model for self-organized systems. In: Intelligent Agent Technology, 2003. IAT 2003. IEEE/WIC International Conference on. pp. 218 – 224 (13-16 2003)
6. Mocan, A., Moran, M., Cimpian, E., Zaremba, M.: Filling the gap - extending service oriented architectures with semantics. In: e-Business Engineering, 2006. ICEBE '06. IEEE International Conference on. pp. 594 –601 (oct 2006)