2005 Pacific Rim Workshop on Intelligent Multi-agents, Kuala Lumpur, 14-16 September 2005, Lecture Notes in Artificial Intelligence, © Springer 2005

Deploying Multi-Agents for Intelligent Aspect-Oriented Web Services

Santokh Singh, John Hosking, John Grundy

Computer Science Dept, University of Auckland, Private Bag 92019, Auckland, New Zealand
{santokh,john,john-g}@cs.auckland.ac.nz

Abstract. The limited description, discovery and integration mechanisms of current web service-based systems have many setbacks that hinder the extension and incorporation of dynamic capabilities into these systems. In this paper we present a novel software architecture called intelligent aspect-oriented web services (IAOWS) which addresses these problems and further improves on this technology by allowing for dynamic look-up and integration. IAOWS use a combination of Aspect-Oriented Multi-Agents and aspectual service descriptors for aspect-oriented web services to cater for more complete and thorough descriptions of services, thus supporting better dynamic discovery of both services and components, and their seamless integration and consumption by clients. We describe our IAOWS architecture and an initial implementation using .NET web services technology to engineer and deploy the Multi-Agents and capture the rich cross-cutting aspects together with their behavior and interaction within our highly distributed system.

1 Introduction

We have been investigating and incorporating the use multi-agents [1, 4] into our research and development on complex and dynamic web service based systems. Web services hold the promise of realizing remote business-to-business integration irrespective of language or platform of the interacting software systems. They achieve this by using proxies to communicate through standard mutually acceptable protocols, the most popular being SOAP over HTTP. They are identified by their unique URIs, whose exposed public interfaces are defined and described in service description documents, the most widely used being Web Services Description Language (WSDL) documents [6, 8, 9]. WSDL uses its W3C XML Schema for defining service interfaces and their disparate endpoints together with data types in an abstract manner, and can map them to any language or middleware interfaces thus providing platform and language independence. These description documents can be discovered by other systems using discovery agencies like the Universal Description, Discovery and Integration (UDDI) [9].

Service requesters (clients) may then interact with the web service in a manner prescribed by its contract, using XML based messages handled by the wsdl proxies. Current web service design techniques and methodologies tend to focus on low level component interface design and implementation. This can result in the development of components whose services are both difficult to comprehend and combine [14, 15].

Aspect Oriented Component Engineering (AOCE) is a complete Component Based Software Development methodology that we have developed, refined and used extensively to better characterize and categorize different systemic cross-cutting capabilities of software components and to reason about inter-component relationships using aspects [15, 16]. Leveraging on AOCE, we have developed a new approach for describing, discovering and integrating web services-based components and have extended the WSDL and UDDI mechanisms to encompass specifications of aspectual properties in web services, characterized these systemic, cross-cutting concerns that impact such systems. These additional aspectual features provide clearer, more enriched and detailed description of web services, thus enabling more comprehensive dynamic discovery, integration and consumption through the use of multi agents.

In the following sections we provide the motivation and significance for our work, give an overview of where and why it is needed with reference to a real situation, then describe the new intelligent aspect-oriented web services (IAOWS) architecture and its specifications We also share our experiences in implementing a prototype .NET-based web services system using our approach. We discuss the strengths and weaknesses of our IAOWS approach and present some directions for future research, particularly in the area concerning multi-agents and web-service based systems.

2 Motivation

Current web service-based systems have many limitations that hamper the incorporation and deployment of dynamic capabilities into these systems. Factors that cause web services to be limited in their description, discovery and integration need to be clearly identified and urgently resolved. The motivation behind this being that web services had held the promise to allow for application to application communication without human intervention, and although it has realized inter-application interaction, so far it has failed to live up to its expectations with regards to automation. In this paper we present a novel software architecture called intelligent aspect-oriented web services (IAOWS) which addresses these problems and further improves on this technology by allowing for dynamic look-up and integration. IAOWS use a combination of Multi-Agents and aspectual service descriptors for aspect-oriented web services to cater for more complete and thorough descriptions of the services, thus supporting better dynamic discovery and seamless integration.

Multi-Agents in web services are semi-autonomous computer programs that employ artificial intelligence techniques to carry out specific tasks, for instance they can assist in the discovery and integration of useful services. These agents can learn through example-based reasoning and are able to improve their performance over time [4, 25]. Multi-Agents can inhabit the complex, dynamic environments of distributed systems, and they can be programmed to sense and act autonomously in these environments. These software robots can think and will act on behalf of a user to realize a set of goals or tasks. We use agents to help fulfill their growing need in our increasing functional, flexible, and highly distributed autonomous web services systems. Uses for these intelligent agents here

include carrying out self-contained tasks, operating semi-autonomously, and communicating between the user and systems resources [3, 16].

A novel Component Based Software Development methodology called AOCE was used because it produces more reusable, scalable, understandable and maintainable software and subsystems compared to existing methodologies. Current CBSD approaches, such as the Architecture Based Component Composition Approach (ABC)[2], TopCoder^{R}, Select Perspective^{TM} [7], OMG's Model Driven Architecture (MDA) [22], and Catalysis^{TM} [12], attempt to use the best approaches from existing traditional software development methodologies including utilizing the power of community-based development, but have not sufficiently addressed the all-important issues of code and designs reusability, understandability, scalability and cross-cutting concerns. They tend to focus on low level features of components rather than component requirements and inter-component relationships, making the components hard to understand, refactor or integrate with each other.

The most comprehensive and one that has a wide range of tool-support is OMG's MDA. It defines software based on UML models, relying on base models called Platform-Independent Models that specify business functionality and behavior in a technology independent manner. An intermediate model called the Platform-Specific Model (PSM) reflects non-business, computing-related details, for instance those affecting performance and resource utilization. The PSM is added to the Platform-Independent Model by the web services' architects and can be used to generate software components. Though it can be applied to model and develop large web service-based systems, it has several setbacks, these include the huge amount of work required to meticulously get all the models and designs of large systems correct, right down to minute details because code is to be generated from them. It is always necessary to improve and tweak the generated code and if mistakes were made in the designs/models the errors will flow through to the generated code requiring further refactoring and debugging. Furthermore, the models and implementations place great emphasis on lower-level systemic features, and this compounds to the inefficiencies introduced and makes the components produced by using OMG's MDA difficult to understand, reconfigure or reuse.

In the Architecture Based Component Composition Approach (ABC)[2] methodology, it is proposed to use Software Architectures (SA) to compose prefabricated components to solve the key issue of component-based reuse. The downside is that SA provides a top-down approach to realizing component-based reuse, but doesn't pay enough attention to the refinement and implementation of the architectural descriptions, thus it is not fully able to automate the transformation or composition to form an executable applications. Though the name ABC may sound simple but in effect it is a complicated CBSD methodology with many strict development rituals and rules to abide by. For developers who are not familiar with ABC or were not involved in the initial development of the particular software, it will pose as a very challenging and uphill task for them as they try to decipher a multitude of architectural diagrams and designs obtained from different phases that are not exactly helpful for reuse, refactoring, maintenance or scaling purposes.

To further complicate matters, TopCoder[R], which is a commercially used CBSD technique, not only relies heavily on the lower-level features of software, it also specifies a cycle of four distinct stages to emphasize these features. These stages are the software's specification, architecture/design, development/testing, and certification, and if any phase were to fail an acceptance test, the phase is restarted all over again from the beginning. As can be seen, this methodology can be tedious. It also focuses on lower-level features of the component/system thereby making designs and implementations hard to understand at abstract levels or during refactoring. Higher level component descriptors such as persistency, distribution, security, performance, transaction processing are not taken into account in any of its four phases. These high-level features are important for understanding and combining systemic components and their functionalities, especially in complex and distributed systems like web services. All the CBSD approaches listed here [2, 7, 12, 22, 24] cannot provide proper support to develop software components for the dynamic discovery and integration of web services because they do not have proper descriptors and formal mechanisms to take these high level features into account, nor do they address critical cross-cutting concerns in these highly distributed systems.

In our earlier work [5] we had laid down the foundations for realizing automation in web services by developing aspect-oriented web services using the AOCE methodology. In that research, we had developed Aspect-Oriented Web Services (AOWS) that had better and more comprehensive descriptors in its exposed interfaces within its Aspect-Oriented WSDL (AOWSDL) documents. This even allows for whole components to be located and consumed by clients instead of just individual operations. Each component has a clearly defined and interrelated set of exposed APIs and by using the whole component we are able to carry out a whole series of related tasks to satisfy a particular goal. Furthermore, the aspectual elements [5] within the AOWSDL service description document enables more complete and accurate discovery of required services. To achieve maximum benefit from our earlier system we refactored and incorporated aspect-oriented multi-agents into our AOWS and called the resulting system IAOWS. The objective here is to enable the realization of more accurate dynamic discovery and integration in our web services model through the use of multi-agents.

3 Intelligent Aspect-oriented Web Services

IAOWS uses the concept of multi-agents and aspects, in this case aspects that impact on different parts of not only the web services, but also the agents. Figure 1 below shows an example from the prototype travel planner system that we developed based on IAOWS. Using discovery agents the client/requester looks-up various services from a registry (1). We deployed Discovery Agents to coordinate with the client's Requesting Agents to search the repository of the AOUDDI, and return results best matching the web service descriptions requested for, including descriptions for their components, aspects, aspect details and provided/required aspectual features

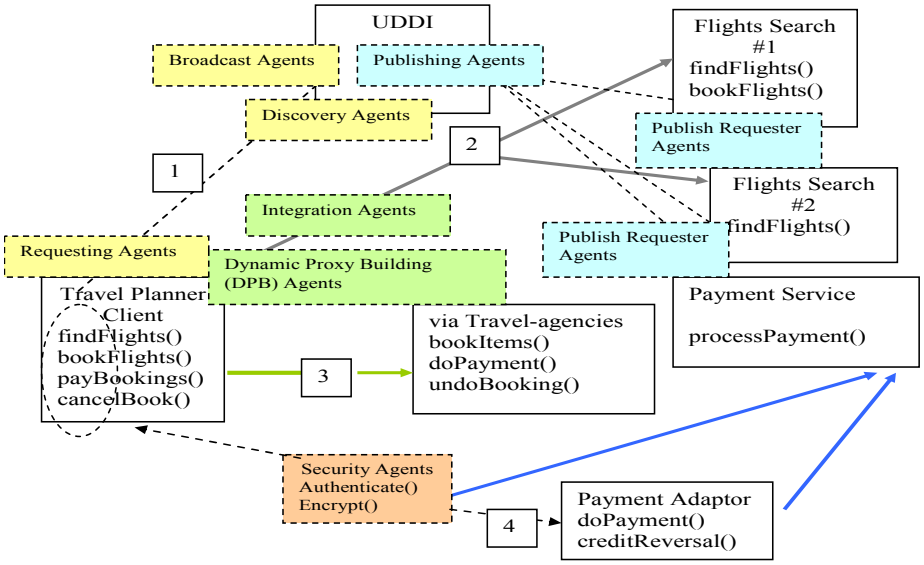

Figure 1. Example of web-service based travel planner utilizing multi-agents.

As shown, flight searches for clients are performed by dynamically integrating with various discovered flight service providers (2) using the integration and proxy building agents. Bookings can be made directly or through agents (3), and if required payment subsequently made through a web-service based allowable mode of payment (4), this series of transactions can be verified by security agents. The Travel Agencies (3) are used as a back-up manual measure for those who do not have the time to search, book etc., and are more comfortable paying others to do these activities for them. Security issues handled by security agents may include a need for user authentication and data encryption/decryption. In specifying client needs and web services providing them, we need to specify these security requirements, and the relevant Multi-Agents will interact, coordinate and negotiate with each other to produce an optimal solution. Aspectual constraints and their required/provided properties are used in testing and validating any discovered service.

To support better dynamic discovery, integration and subsequent consumption of services in web-service based systems, we designed and developed Intelligent Aspect-oriented Web Services (IAOWS) using Multi-Agents. This research further extends our AOCE work in which we developed extensions to the object component model to support component design, de-coupled implementation and run-time discovery and integration using component aspects [5, 6, 16]. Component aspects are cross-cutting concerns impacting on components, including persistency management, distribution, security, transaction processing and resource use. Components provide capabilities to others or require services from them across these different system aspects. Aspect details capture

functional and non-functional properties and allow design-time reasoning and run-time component description and adaptation.

Key aspects that multi-agents use when discovering web services to interact with include security model, transaction management, performance measures for operations, and fault and exception-handling approaches. As such, when building web services we may describe data persistency approach, database transactional behaviour for operations, resource utilization, communications infrastructure, monitoring and logging, etc. During discovery and integration, we may need to locate adaptors, transaction managers, and security managers, and compose (or orchestrate) services. We aim to better support this range of activities when designing, implementing and deploying web services using IAOWS. We have developed a model of IAOWS-based systems, together with a variety of multi-agents, and proof-of-concept implementation of the model with .NET web services.

4 Multi-Agents of the IAOWS

We used a variety of agents, each assigned clear and specific tasks, to enable the dynamic discovery and integration of the web services to be realized, including consuming the services. We had to be very careful to assign tasks to the correct agents that are most appropriate to handle them and to ensure that there were no overlapping tasks [25]. We also had to ensure that the agents were communicating and coordinating with their appropriate counter-parts/subsystems. These agents were run using their own separate threads asynchronously so that they did not hold up processing time and can compete with each other for resources on a first-come-first-served basis.

Figure 2 below shows the architecture of our IAOWS. As depicted here, IAOWS uses the concept of multi-agents and aspects, in this case aspects that impact on different parts of web services that can be captured and utilized by the agents. We had used our novel AOCE methodology to develop IAOWS because the other current CBSD methodologies discussed earlier neither identify nor address the issues of these cross-cutting concerns called aspects. Furthermore, they do not produce aspect-oriented components which are the highly reusable and understandable building blocks of our IAOWS and its various subsystems. Even our multi-agents are composed of aspect-oriented components to make them more reusable and efficient, and our IAOWS as a whole more modularized.

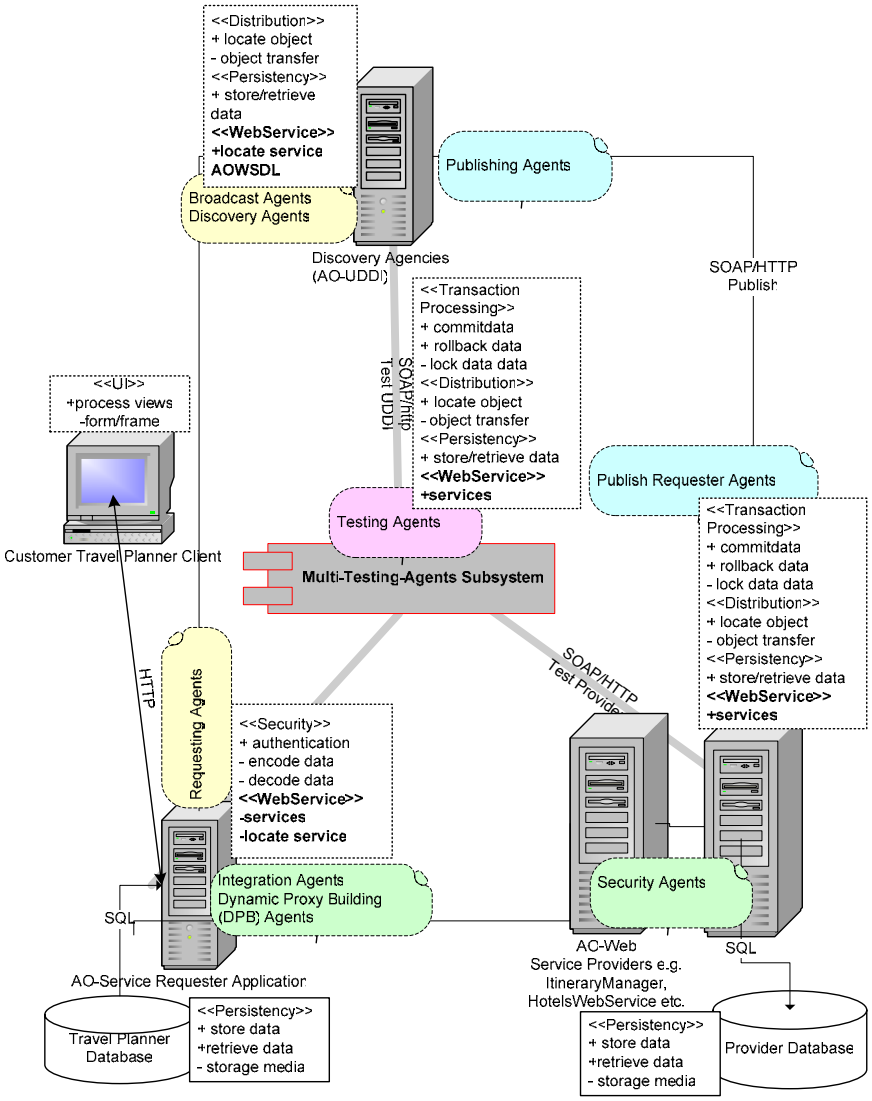

Figure 2. The architecture of Intelligent Aspect-oriented Web Services

In this figure, Aspect-oriented web service providers use their own Publish Requester Agents (PR Agents) to publish their services to the AOUDDI. PR Agents do this by transmitting and depositing the unique AOWSDL document of their respective provider to the registry. Publishing agents in the AOUDDI coordinate with the PR agents and if the

publishing is successful, issue the PR agents with a unique identity number called an ID_Publishing number. Publishing agents in the AOUDDI will first check to see whether the document already exists in their repository and if so, whether it is a duplicate copy or an updated version. No new ID_Publishing number is issued in either of these cases. But if it was a new service the publishing agent will automatically generate and issue a new ID which it will also store in its database together with the AOWSDL document. If there is an exact copy already registered in its repository, then the redundant AOWSDL document that was submitted is discarded and no further action needs to be taken by the publishing agents. All actions taken and processing done by the publishing agents is stored in a cache so that it can be indexed first and action taken almost instantaneously. This makes the agents more efficient and can safeguard against multiple attacks by unscrupulous providers making repeated publications to overload the AOUDDI.

If it is an updated AOWSDL version, the publishing agents will call the broadcast agents in the AOUDDI to broadcast to all requesters via their requesting agents registered with the AOUDDI that a new version of the AOWSDL document from a particular service provider is available. The requesting agents will verify with their integration agents whether the particular web service is currently being consumed. If the reply is positive, the integration agents will give a complete list of services used and their required aspectual features to the requesting agents. The requesting agents will then communicate with the discovery agents in the AOUDDI to verify whether the required services are still available from the provider. It does multiple XML queries on the discovery agents which include requesting for particular aspects, the details and provided/required properties. The discovery agents resort to case based reasoning to answer the queries of the requestors. The discovery agents also have efficient parsers that parse the whole AOWSDL document and pull out all the information within the document and store it in Hash Tables for detailed look-up purposes. All communication between the distributed and collaborating agents on different machines is done in XML format using the SOAP over HTTP protocol.

If an already consumed but updated provider can still provide the services that the requester needs, then the remote web reference (proxy) [8, 21] in the requester needs to be updated dynamically. The requesting agents will call the integrating agents to update the reference. The integrating agents will instruct the dynamic proxy building (DPB) robot in the requester to dynamically destroy the existing proxy of the web service if the provider is not currently being called to carry out remote processing. The DPB robot then dynamically recreates a new proxy class based on the updated AOWSDL file, adding all the relevant assemblies and functions to it in C#. It then compiles the AOWSDL proxy class into a dynamic link library (.dll) file, saves it in the BIN (.NET's binaries) folder and adds a reference to it for the requester. This completes the task of dynamically updating the proxy class. The DPB robot can then use reflection to dynamically discover and call all methods, with their parameters and properties on this proxy class. The integrating agents are notified that the dynamic proxy is created and these agents in turn pass on the command to the requester so that the proxy can now be used by clients.

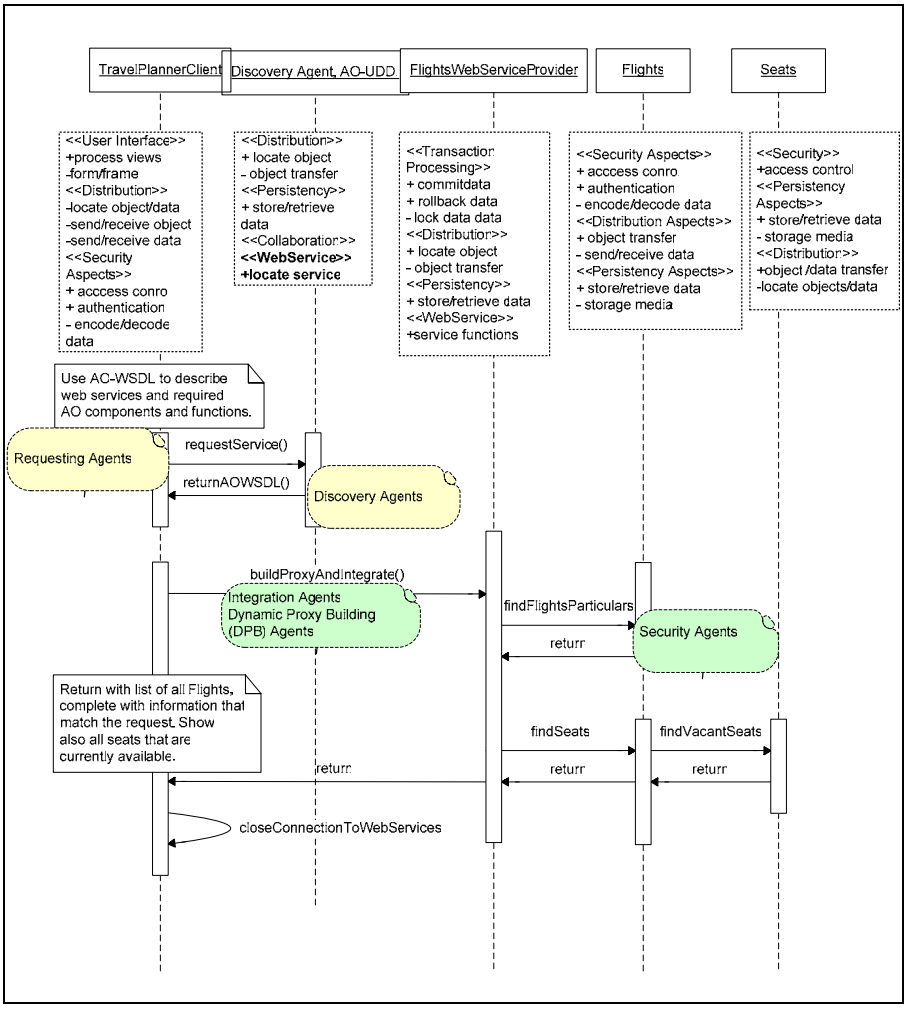

Figure 3. Sequence diagram showing dynamic discovery, integration and consumption of a flights web service using multi-agents

Figure 3 above shows an example of an AO-Sequence Diagram for the collaborative Travel Planner with aspects and Multi-Agents, it describes the sequence of events for searching, integrating and consuming a Flights web service. The requesting agents here request for a flights service for making reservations on flights. Discovery agents do an AI search and return a best matched service based on the request. An interplay of AI agents with the help of aspect-enriched queries and responses together with coordinated effort allows this to be achieved. The Multi-agents shown here follow all the procedures and transactions explained earlier in this section. Shown here also are security agents that only

allow service bindings and interactions with clients that are authorized to use the services. This is achieved by inserting secret encoded keys [6] for access to the web service by the client in its XML requests that are deciphered by the security agents. Access is only allowed if the correct code is used by the client, and this kind of transactions are for instance used to authorize staff to edit databases entries etc. that normal customers do not have permission to do.

5 Implementation

We designed and developed a prototype collaborative Travel Planner based on the IAOWS model of deploying Multi-Agents in a remotely connected server that can dynamically discover and integrate with relevant aspect-oriented web service providers so that users can use it to plan and make bookings for various itinerary items for their travel or holidays. Figure 4(a.) below shows a section of the GUI of this web-based Travel Planner developed using AOCE. It shows the web form (a .aspx type file) of the application used to search and subsequently book and make payments for flights to particular destinations. We also implemented a trimmed down version of the GUI containing all its functionalities in a smart device application shown in Figure 4(b.).

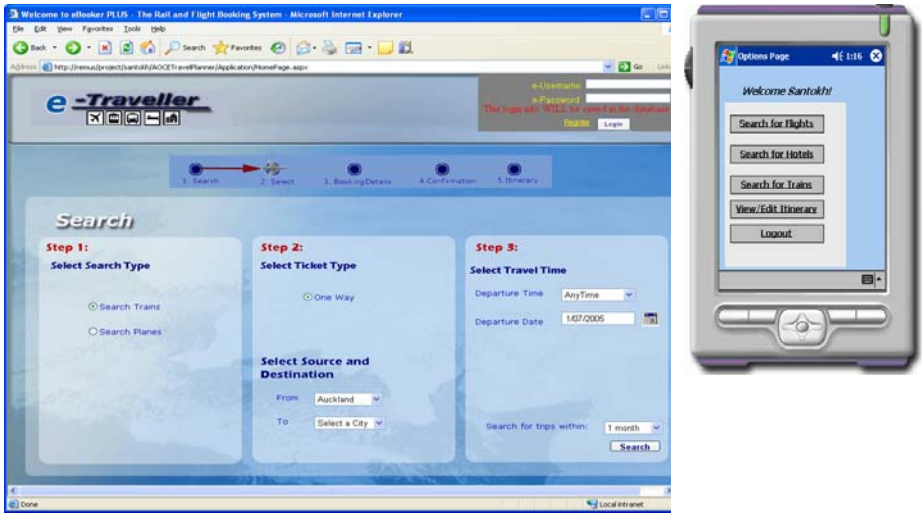

Figure 4. Travel planner applications (a.) web based in PC (b) smart device application that interacts with the web services.

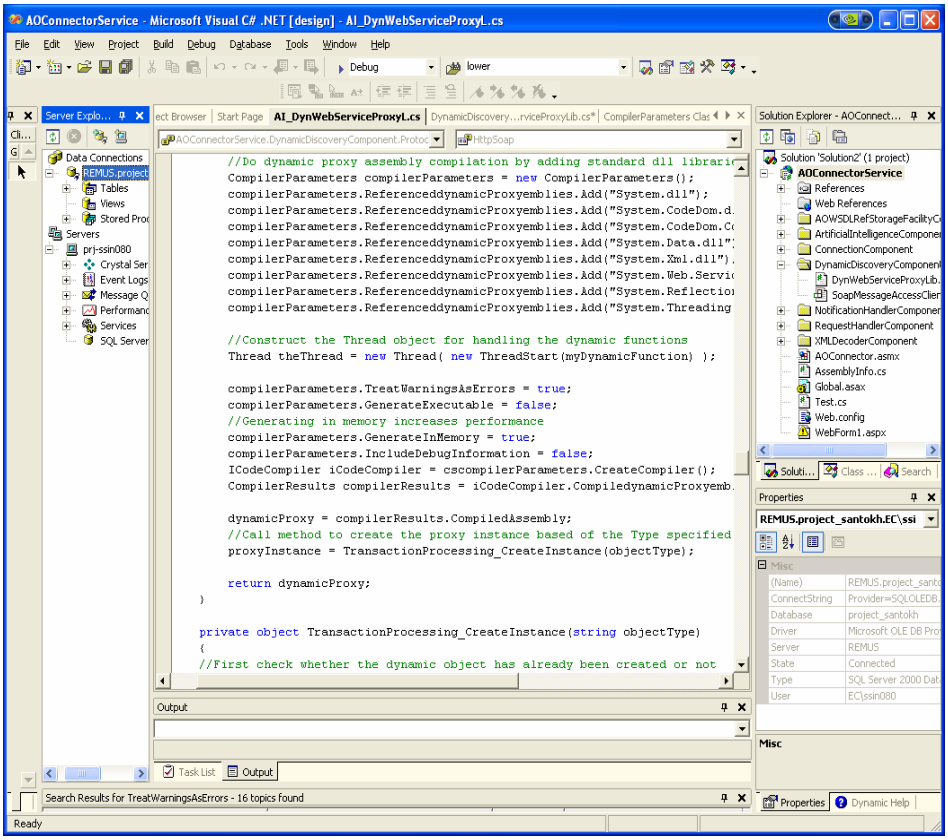

Figure 5 C# Code of the dynamic proxy building (DPB) robot in the requesters.

Figure 5 shows a sample of the code written in C# of the dynamic proxy building (DPB) agent in the requesters. This agent is used to dynamically create or update a web service proxy based on an AOWSDL file found to be suitable through the coordinated effort of the requesting agents in the clients and discovery agents in the AOUDDI. Integrating agents instruct the DPB robot in the requester to dynamically destroy the 'old' proxy (if any) first before it creates a new proxy class based on the discovered service document. As can be seen, all the relevant assemblies and functions are added to the proxy first. The AOWSDL proxy class is then generated, compiled and referenced by the DPB agent so that it can be used by the requester. The integrating agents dynamically create instances of this proxy to enable remote execution of operations on it using the SOAP/HTTP as the transport protocol.

All agents are placed within well defined aspect-oriented components [5, 15, 16] so that they can be easily reused and to achieve better modularity. This high level of modularity allowed us to rapidly and accurately refactor, update and test our AI

algorithms, especially algorithms for conducting searches and case based reasoning purposes. Also aspect-oriented components address cross cutting concerns [19, 20] and are better characterized and categorized. Implementation was done entirely in C# using Microsoft's Visual Studio and the .NET Framework [8, 21] so that we could concentrate on the core issues involving the architecture, design and deployment of Multi-Agents within IAOWS without the strain of learning and debugging in a multitude of languages.

6 Discussion

We had to carefully plan and limit the number and different kinds of agents in our IAOWS system to an optimal and controllable number [11, 25] so that the series of tasks executed by each type of agent is clearly defined and not overlapped with other types of agents. We had to define the tasks precisely so that when it needs to be executed we know exactly which agent to call into action. Since we already had experience designing and developing an earlier prototype of AOWS (without the multi-agents) based on our earlier research, this extension was not too difficult as we already had the technical knowledge and expertise of how the distributed system works. We reused the code and aspect-oriented components from the earlier prototype, and refactored where necessary, and this made our development based on AOCE techniques more efficient and effective.

We used AI algorithms e.g. CBR in discovery agents mining the AOUDDI repository, and A* search algorithms to choose the best web service that meets the search criteria composed of aspect-enhanced multiple queries. As the number or registered web services in the repository of the registry grew, searches became slower because of the huge amount of information that had to be processed based on the criteria the agents had to match and satisfy. We tried using other algorithms like Best-First Search, Greedy Search and Means-Ends Analysis, but we abandoned them as the results obtained through these techniques were not as good as the A* Search which merges two heuristic functions into one superior function and this can satisfactorily process aspectual queries.

We also discovered that the deployment of agents increased the modularity in our software systems. All our sub-systems have become more lightweight as we have extracted a multitude of key operations/components from our requesters, providers and AOUDDI subsystems and placed them in Multi-Agents. In our updated system that we presented in this paper, we now just need to call these agents to carry out their specific tasks in the software. As such, our IAOWS system and its aspect-oriented components are now easier to reuse and refactor, making the overall system more maintainable and scalable.

In our ongoing and future work we are looking into avenues of extending and deploying these agents in a semantic aspect-oriented web services system that we are currently in the initial stages of designing and formulating. We will also add in other coordinating agents here to carry out any additional tasks involved due to the introduction of the new features. We believe that the semantics and aspects will give the agents their full power and capabilities to carry out more comprehensive and accurate dynamic discovery, integration and consumption of web services within our IAOWS framework.

7 Summary

This extension to our earlier work where we had developed Aspect-Oriented Web Services is a particularly significant phase in the design and development of web-service based systems that can support automation in the area of dynamic discovery, integration and subsequent consumption by clients through the use of Multi-Agents and aspectual features. It allows us to come nearer at realizing the dream that web services can indeed cater for dynamic application to application communication without human intervention. The Multi-Agents deployed here not only addressed the issues that hampered dynamic look-up and integration of web-service based systems, they also made such systems more modular, maintainable, reusable and scalable.

Acknowledgements

We gratefully acknowledge the helpful comments of the anonymous referees on an earlier draft of this paper. This work has been supported in part by the New Zealand Foundation for Research, Science and Technology and the University of Auckland Research Committee.

References

1. Secq, Y., Routier, J., Mathieu, P., Dynamic Organization of Multi-Agent Systems, PRIMA '02 Tokyo, Japan.
2. Mei, H,, ABC: Supporting Software Architectures in the Whole Lifecycle, Proceedings of the Second International Conference on Software Engineering and Formal Methods (SEFM'04), IEEE.
3. Keller, R. M., Wolfe, S. R., Chen, J. R., Rabinowitz, J. L., Mathe, N., A Bookmarking Service for Organizing and Sharing URLs; Proceedings of the Sixth International WWW Conference, Santa Clara, CA, 1997.
4. Rahwan, I., Graham, C., Sonenberg, L., Supporting Impromptu Coordination Using Automated Negotiation, PRIMA 2004 New Zealand.
5. Singh, S., Grundy, J., Hosking, J.,. Developing .NET Web Service-based Applications with Aspect-Oriented Component Engineering, AWSA'04, Australia.
6. Adams, C., Boeyen, S. UDDI and WSDL extensions for Web service: a security framework, In Proc. 2002 ACM workshop on XML security, Fairfax, VA , 2002.
7. Allen, P., and Frost, S. Component-Based Development for Enterprise Systems: Applying the Select Perspective, Addison-Wesley, 1998.
8. Ballinger, K., .NET Web Services: Architecture and Implementation, Addison-Wesley, 2003.
9. Cerami, E. Web Services Essentials - Distributed Applications with XML-RPC, SOAP, UDDI & WSDL, Feb 2002, O'Reilly.
10. Colyer A., Clement, A., Large-scale AOSD for Middleware, AOSD 04, ACM.

11. Gómez, M., Plaza, E. Extending Matchmaking to Maximize Capability Reuse. In Proc. Third International Joint Conference in Autonomous Agents and Multiagent Systems (AAMAS'04), ACM.
12. D'Souza, D.F. and Wills, A.C. Objects, Components and Frameworks with UML, The Catalysis Appproach, Addison-Wesley, 1999.
13. Gannod, C., Bhatia, S. Facilitating Automated Search for Web Services, In Proc. IEEE International Conference on Web Services, ICWS'04, IEEE.
14. Grundy, J.C. and Hosking, J.G., In Engineering plug-in software components to support collaborative work, S-P&E, 2002; vol. 32, pp. 983-1013.
15. Grundy, J. Multi Perspective Specification, Design and Implementation of Software Components using Aspects, In Int. J. Soft. Eng. and Knowledge Eng. Vol. 10, No. 6 (2000), pp. 713-734, World Scientific.
16. Grundy, J. and Ding, G. Automatic Validation of Deployed J2EE Components Using Aspects, In Proc. 2002 IEEE International Conference on Automated Software Engineering, Edinburgh, UK, IEEE CS Press.
17. Hausmann, J.H.H., Heckel, R., Lohmann, M., Model-based Discovery of Web Services, In Proc. ICWS'04.
18. Katara, M., Katz, S., Architectural Views of Aspects*, In Proc. AOSD 2003, Boston, MA USA, ACM 2003.
19. Kiczales et al, Aspect-oriented Programming, In Proc. the 1997 European Conf. on Object-Oriented Programming, Finland (June 1997), Springer-Verlag, LNCS 124.
20. Lieberherr, K. Connections between Demeter/Adaptive Programming and Aspect-Oriented Programming (AOP), http://www.ccs.neu.edu/home/lieber/, 1999.
21. Microsoft, Visual Studio and .NET, http://www.microsoft.com/net/, 2003, Microsoft..
22. Siegel, J. Using OMG's Model Driven Architecture (MDA) to Integrate Web Services, http://www.omg.org/.
23. Stearns, M., Piccinelli, G., Managing Interaction Concerns in Web-Service Systems, Proc. 22nd Int. Conf. on Distributed Computing Systems Workshops, pp. 424.
24. Vitharana, P., Mariam, F., and Jain, H., Design, Retrieval, And Assembly in Component-based Software Development, CACM, vol. 46, no. 11, Nov. 2003.
25. Shen, J., Weber, I., Lesser, V., OAR: A Formal Framework for Multi-Agent Negotiation, American Association for Artificial Intelligence, AAAI 2005