Distributing Functionalities in a SOA-Based Multi-agent Architecture

Dante I. Tapia, Javier Bajo, and Juan M. Corchado

Departamento Informática y Automática
Universidad de Salamanca
Plaza de la Merced s/n, 37008, Salamanca, Spain
{dantetapia, jbajope, corchado}@usal.es

Abstract. This paper presents how functionalities are distributed my means of FUSION@, a SOA-based multi-agent architecture. FUSION@ introduces a new perspective for constructing multiagent systems, facilitating the integration with service-oriented architectures, and the agents act as coordinators and administrators of services. FUSION@ makes use of several mechanisms for managing and optimizing the services distribution. The results obtained demonstrate that FUSION@ can efficiently distribute functionalities in dynamic scenarios at execution time.

Keywords: Multi-Agent Systems, Services Oriented Architectures, Distributed Computing.

1 Introduction

Multi-agent systems are very appropriate for resolving problems in a distributed way [9]. Agents have a set of characteristics, such as autonomy, reasoning, reactivity, social abilities, pro-activity, mobility, organization, etc. which allow them to cover several needs for dynamic environments, especially ubiquitous communication and computing and adaptable interfaces. Agent and multi-agent systems have been successfully applied to several scenarios, such as education, culture, entertainment, medicine, robotics, etc. [6], [15]. Moreover, the continuous advancement in mobile computing makes it possible to obtain information about the context and also to react physically to it in more innovative ways [9]. Nevertheless, complex systems need higher adaptation, learning and autonomy levels than pure BDI model [3]. This can be achieved by modelling the agents' characteristics [21] to provide them with mechanisms that allow solving complex problems and autonomous learning [7]. However, excessive complex mechanisms (e.g. data mining, genetic algorithms, indexing, learning techniques, visualization, etc.) can cause malfunctioning and crashes in multi-agent systems developed with current technologies such as agent platforms. This is because developers tend to integrate all functionalities inside the agents' internal structure, creating agents with high computational requirements.

The *Flexible User and ServIces Oriented multi-ageNt Architecture* (FUSION@) [18] tries to solve this problem. One of the most important characteristics in FUSION@ is the use of intelligent agents as the main components in employing a service oriented approach, focusing on distributing the majority of the systems'

Y. Demazeau et al. (Eds,): 7th International Conference on PAAMS'09, AISC 55, pp. 20–29.
springerlink.com

functionalities into remote and local services and applications. The architecture proposes a new and easier method of building distributed multi-agent systems, where the functionalities of the systems are not integrated into the structure of the agents; rather they are modelled as distributed services and applications which are invoked by the agents acting as controllers and coordinators. This approach optimizes usability and performance because it can be obtained lighter agents in terms of computational load.

FUSION@ has been previously presented in related papers [18], however, this paper focuses on describing how FUSION@ implements several mechanisms for optimizing and managing functionalities and resources in dynamic environments. Through these mechanisms, FUSION@ facilitates the development of distributed multi-agent systems and the agents have the ability to dynamically adapt their behaviour at execution time. FUSION@ provides an advanced flexibility and customization to easily add, modify or remove applications or services on demand, independently of the programming language. It also formalizes the integration of applications, services, communications and agents. The proposed approach has been applied to a real scenario to evaluate the performance in a multi-agent system for monitoring Alzheimer patients.

In the next section, the specific problem description that essentially motivated this research is presented. Section 3 describes the main characteristics of the FUSION@ architecture and briefly explains the mechanisms for distributing and optimizing functionalities and resources. Section 4 presents the results and conclusions obtained after testing the architecture in a real scenario.

2 Problem Description and Background

Excessive centralization of services negatively affects the systems' functionalities, overcharging or limiting their capabilities. Classical functional architectures are characterized by trying to find modularity and a structure oriented to the system itself. Modern functional architectures like Service-Oriented Architecture (SOA) consider integration and performance aspects that must be taken into account when functionalities are created outside the system. These architectures are aimed at the interoperability between different systems, distribution of resources, and the lack of dependency of programming languages [5]. Services are linked by means of standard communication protocols that must be used by applications in order to share resources in the services network [1]. The compatibility and management of messages that the services generate to provide their functionalities is an important and complex element in any of these approaches.

One of the most prevalent alternatives to these architectures is agents and multi-agent systems technology which can help to distribute resources and reduce the central unit tasks [1] [19]. A distributed agents-based architecture provides more flexible ways to move functions to where actions are needed, thus obtaining better responses at execution time, autonomy, services continuity, and superior levels of flexibility and scalability than centralized architectures [4]. Additionally, the programming effort is reduced because it is only necessary to specify global objectives so that agents cooperate in solving problems and reaching specific goals, thus giving the systems the ability to generate knowledge and experience.

Agent and multi-agent systems combine classical and modern functional architecture aspects. Multi-agent systems are structured by taking into account the modularity in the system, and by reuse, integration and performance. Nevertheless, integration is not always achieved because of the incompatibility among the agents' platforms (e.g. JADE agents with RETSINA or OAA agents). The integration and interoperability of agents and multi-agent systems with SOA and Web Services approaches has been recently explored [1] [9]. Some developments are centred on communication between these models, while others are centred on the integration of distributed services, especially Web Services, into the structure of the agents [11] [14] [16]. Although these developments provide an adequate background for developing distributed multi-agent systems integrating a service oriented approach, most of them are in early stages of development, with little systems (if any) developed upon them so it is not possible to actually know their potential in real scenarios. Please refer to [18] for a detailed comparison of these approaches.

3 FUSION@, A SOA-Based Multi-agent Architecture

FUSION@ [18] is a modular multi-agent architecture, where services and applications are managed and controlled by deliberative BDI (Belief, Desire, Intention) agents, [3], [13], which mainly follows the principles of the THOMAS architecture [12]. Deliberative BDI agents are able to cooperate, propose solutions on very dynamic environments, and face real problems, even when they have a limited description of the problem and few resources available [2], [8]. There are different kinds of agents in FUSION@, each one with specific roles, capabilities and characteristics. This fact facilitates the flexibility of the architecture in incorporating new agents. FUSION@ defines four basic blocks which provide all the functionalities of the architecture.

- Applications. Represent all the programs that can be used to exploit the system functionalities. They can be executed locally or remotely, even on mobile devices with limited processing capabilities, because computing tasks are largely delegated to the agents and services.
- Agents Platform. This is the core of FUSION@, integrating a set of agents, each one with special characteristics and behaviour. An important feature in this architecture is that the agents act as controllers and administrators for all applications and services, managing the adequate functioning of the system, from services, applications, communication and performance to reasoning and decision-making.
- Services. They are the bulk of the functionalities of the system at the processing, delivery and information acquisition levels. Services are designed to be invoked locally or remotely. Services can be organized as local services, web services, or even as individual stand alone services.
- Communication Protocol. This allows applications and services to communicate directly with the agents' platform. The protocol is completely open and independent of any programming language. This protocol is based on SOAP specification to capture all messages between the platform and the services and applications [5]. All external communications follow the same protocol, while the communication among agents in the platform follows the FIPA Agent Communication Language

(ACL) specification. Applications can make use of agents platforms to communicate directly (using FIPA ACL specification) with the agents in FUSION@, so while the communication protocol is not needed in all instances, it is absolutely required for all services.

There are pre-defined agents that provide the basic functionalities of FUSION@. *CommApp Agent* is the agent is responsible for all communications between applications and the platform. It manages the incoming requests from the applications to be processed by services. It also manages responses from services (via the platform) to applications. All messages are sent to *Security Agent* for their structure and syntax to be analyzed. *CommServ Agent* is responsible for all communications between services and the platform. The functionalities are similar to *CommApp Agent* but backwards. *Admin Agent* signals to *CommServ Agent* which service must be invoked. All messages are sent to *Security Agent* for their structure and syntax to be analyzed. This agent also periodically checks the status of all services to know if they are idle, busy, or crashed. *Directory Agent* manages the list of services that can be used by the system. For security reasons [17], FUSION@ does not include a service discovery mechanism, so applications must use only the services listed in the platform. However, services can be added, erased or modified dynamically. There is information that is constantly being modified: the service performance (average time to respond to requests), the number of executions, and the quality of the service (QoS). This last data is very important, as it assigns a value between 0 and 1 to all services. All new services have a quality of service (QoS) value set to 1. This value decreases when the service fails (e.g. service crashes, no service found, etc.) or has a subpar performance compared to similar past executions. QoS is increased each time the service efficiently processes the tasks assigned. *Supervisor Agent* supervises the correct functioning of the other agents in the system. *Supervisor Agent* periodically verifies the status of all agents registered in the architecture by sending ping messages. If there is no response, the Supervisor agent kills the agent and creates another instance of that agent. *Security Agent* analyzes the structure and syntax of all incoming and outgoing messages. If a message is not correct, the *Security Agent* informs the corresponding agent (*CommApp* or *CommServ*) that the message cannot be delivered. This agent also directs the problem to the *Directory Agent*, which modifies the QoS of the service where the message was sent. *Admin Agent* decides which agent must be called by taking into account the QoS and users' preferences. Users can explicitly invoke a service, or can let the *Admin Agent* decide which service is best to accomplish the requested task. This agent also checks if services are working properly. It requests the *CommServ Agent* to send ping messages to each service on a regular basis. If a service does not respond, CommServ informs *Admin Agent*, which tries to find an alternate service, and informs the *Directory Agent* to modify the respective QoS. *Interface Agent* was designed to be embedded in users' applications. *Interface agents* communicate directly with the agents in FUSION@ so there is no need to employ the communication protocol, rather the FIPA ACL specification. The requests are sent directly to the *Security Agent*, which analyzes the requests and sends them to the *Admin Agent*. These agents must be simple enough to allow them to be executed on mobile devices, such as cell phones or PDAs. All high demand processes must be delegated to services.

3.1 Services Management

FUSION@, and in particular the *Admin Agent*, employs a mechanism composed of a set of techniques that allows the architecture to select the most appropriate service to meet a request at any given time. The mechanism to assign the most appropriate service to respond to this request begins when a new request is received, taking into account the following parameters: the QoS value; the user preferences; and the estimated delivery time. The first two parameters are set in advance so it is not necessary to calculate by the *Admin Agent*. The execution time is estimated using a RBF (Radial Basis Function) Neural Network. The reason for using this type of network is the speed in the training phase, compared to a Multi Layer Perceptron (MLP). The neural network is made up of three layers: the input layer, an intermediate/hidden layer and an output layer. The number of neurons in the input layer is defined by the number of entries (i.e. parameters) to each service. In the case of arrays, each element counts as an entry to the service. The number of neurons in the middle layer is determined dynamically, making a cross-training and validation. The cross-validation is done through the method GCV (Generalized Cross-Validation) [19]. The variation in the number of neurons in the middle layer is made by the following algorithm.

First are initialized the list of errors $e = \{\}$ and the list of angles $\alpha = \{\}$. Then the training is conducted with a neuron in the middle layer and the training value is stored in e_1 in the values list $e = e \cup e_1$. Next, the number of neurons in the intermediate layer is initialized. If there is no prior training, it is initialized to $4n+1$ where n is the number of neurons in the input layer. However, if there is a prior training, it is initialized to $2n+1$, where n is the number of neurons of the prior training. r is the number of neurons in the current layer. The training is conducted for r intermediate neurons and the error in e_r $e = e \cup e_r$ is stored. Subsequently, the training is done for $r/2$ neurons and the error in er/2 $e = e \cup e_{r/2}$ is stored. The lines that pass through the points $r_{1,r/2} = (e_r e_{r/2})$ and $r_{1,r/2} = (e_1 e_{r/2})$ are calculated and the angle $\alpha_{r/2}$ formed by these lines is obtained. The angle $\alpha_{r/2}$ is introduced in the list of angles $\alpha = \alpha \cup_{r/2}$. If there is only one angle on the list of angles $\#\alpha = 1$ then the value of r is set to $r/4$ and the training is restarted. If $\#\alpha = 2$, being α_i the other existent value, then: If $i<r/2$ then $r = r/2$ and the training is restarted; else $r = r/2+r/4$ and the training is restarted. But selecting the two adjacent values to the left and right of $\alpha_{r/2}$ denoted by α_i and α_j so that:

- If $\alpha_i > \alpha_j$
  i. If $j=r/2+1$ the value of neurons is set to r
  ii. Else, set the new value of $r=r/2+(j-r/2)/2$
- Else
  i. If $i=r/2-1$ the value of neurons is set to r
  ii. Else, set the new value of $r=r/2-(r/2-i)/2$

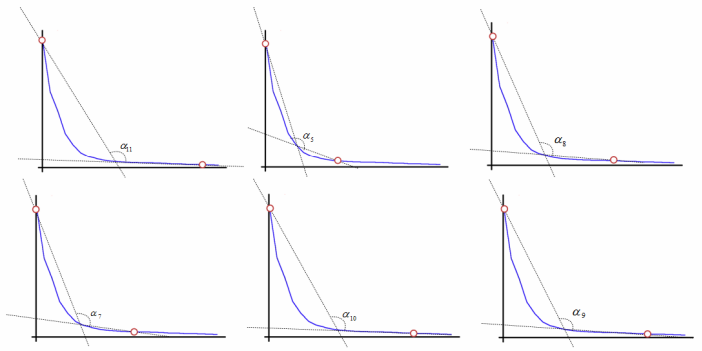

Fig. 1. Progression of the number of neurons in the hidden layer

Finally, the training is carried out and the value of neurons in the intermediate layer is set to *r*. Figure 1 shows the progress of the algorithm searching the lower angle α. The number of neurons in the middle layer is shown in the X axis and the cross validation is shown in the Y axis. The values correspond to a simulation, but for a real case there is only necessary to calculate the values that are obtained in the sets listed next. The sets of values studied at every moment are (23, 11, 1) (11, 5, 1) (17, 8, 1) (14, 7, 1) (20, 10, 1) (18, 9, 1). The end result is given by (18, 9, 1) therefore, the number of neurons in the final layer is 9. The output layer consists of a single neuron, whose output value is the time prediction.

Learning consists of an unsupervised learning phase for the middle layer and other supervised learning phase for the output layer. A cross-validation is carried out at the training phase by varying the number of neurons in the intermediate layer in case of no new results obtained. First, the training of the middle layer and the output layer are carried out. The retraining of the network is carried out when the average time is *k* times larger or smaller than the estimated time for the last *n* executions. These values are defined in advance for each system. A cross-validation is done through the GVC method in order to determine the completion of the training.

The available services can be assigned once the execution time of the requests in the queue is estimated. The assignation tries to maximize the performance in terms of the time required to respond to all requests. This assignation must be efficient and dynamically adaptable because it must be carried out in execution time. For this reason, a FIFO model has been chosen for each service. Each service has a queue associated. There is a common queue where all requests are queued and managed by the *Admin Agent*. The allocation to the queue of each service is done taking into account the requests assigned to each queue and the estimated performance given by (1). The request is assigned to the queue in order to minimize the cumulative performance and the estimation time.

$$r_i = (1 - QoS_i) \cdot t \cdot (1 - p_i) \tag{1}$$

Where: r_i is the estimated performance of the service *i*; QoS_i is the quality of service *i* which corresponds to a value between 0.00 and 1.00; T is the estimated time of duration; and P_i are the preferences for the service *i* with a value between 0 and 1. Being *n*

the number of services, R_i the cumulative performance for the queue i, a new request s with performance estimations for the service j r_{ij} will be assigned to the queue R_i when satisfying the following condition. If the value of the preference is equal to 1, only the queues with that value for the preference are taken into account.

$$\min\{R_1 + r_{i1},..., R_n + r_{in}\} \quad (2)$$

Assuming that it is assigned to the k queue, the new cumulated performance for the k queue will be R_k+r_{ik} and the queuing mechanism continues for the next service. Figure 2 shows a representation of the queuing mechanism. There is a primary queue associated with all services. Each request has an estimated performance r_{ij} and is associated with a queue until all queues are occupied.

In this way, it is possible to assign the requests entering into the system to each of the available services. This leads to a better performance and optimization of resources.

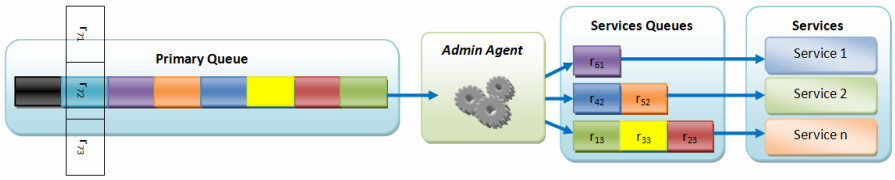

Fig. 2. Services assignation through the queuing mechanism

4 Results and Conclusions

Several tests have been done to demonstrate if the mechanisms in FUSION@ are appropriate to distribute resources and optimize the performance of multi-agent systems. Most of these tests basically consist on the comparison of two simple configurations (System A and System B) with the same functionalities. These systems are specifically designed to schedule a set of tasks using a planning mechanism [6]. System A integrates this mechanism into a deliberative BDI agent, while System B implements FUSION@, modelling the planning mechanism as a service. Table 1 shows an example of the results delivered by the planning mechanism for both systems. An agenda is a set of non organized tasks that must be scheduled by means of the planning mechanism or the planner service. There were 30 defined agendas each with 50 tasks. Tasks had different priorities and orders on each agenda. Tests were carried out on 7 different test groups, with 1, 5, 10, 15, 20, 25 and 30 simultaneous agendas to be processed by the planning mechanism. 50 runs for each test group were performed, all of them on machines with equal characteristics. Several data have been obtained from these tests, focusing on the average time to accomplish the plans. For System B five planner services with exactly the same characteristics were replicated.

Table 1. Example of the results delivered by the planning mechanism

Time	Activity
19:21	Exercise
20:17	Walk
22:00	Dinner

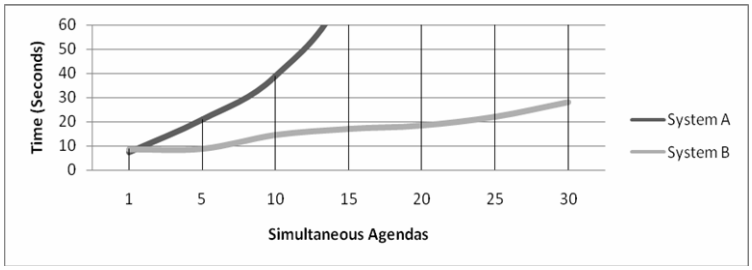

Fig. 3. Time needed for both systems to schedule simultaneous agendas

Figure 3 shows the average time needed by both systems to generate the paths for a fixed number of simultaneous agendas. System A was unable to handle 15 simultaneous agendas and time increased to infinite because it was impossible to perform those requests. However, System B had 5 replicated services available, so the workflow was distributed, and allowed the system to complete the plans for 30 simultaneous agendas. Another important data is that although the System A performed slightly faster when processing a single agenda, performance was constantly reduced when new simultaneous agendas were added. This fact demonstrates that the overall performance of System B is better when handling distributed and simultaneous tasks (e.g. agendas), instead of single tasks.

The FUSION@ architecture proposes an alternative where agents act as controllers and coordinators. The mechanisms implemented in FUSION@ can distribute resources, exploiting the agents' characteristics to provide a robust, flexible, modular and adaptable solution that covers most of the requirements of a wide diversity of projects. All functionalities, including those of the agents, are modelled as distributed services and applications. By means of the agents, the systems are able to modify their behaviour and functionalities at execution time. Developers can create their own functionalities with no dependency on any specific programming language or operating system.

Initial results demonstrate that FUSION@ is adequate for distributing composite services and optimizing performance for multi-agent systems. The *Admin Agent* learns and reason, which facilitates the optimum distribution of tasks and reduces the processing for the rest of the agents in the system. Future work consists on applying this architecture into composite multi-agent systems, as well as extending the experiments to obtain more decisive data from applications that consume multiple services with different capabilities in heterogeneous scenarios.

Acknowledgements. This work has been partially supported by the TIN2006-14630-C03-03 and the IMSERSO 137/07 projects. Special thanks to Tulecom for the technology provided and the know-how supported.

References

1. Ardissono, L., Petrone, G., Segnan, M.: A conversational approach to the interaction with Web Services. In: Computational Intelligence, vol. 20, pp. 693–709. Blackwell Publishing, Malden (2004)
2. Bratman, M.E.: Intentions, plans and practical reason. Harvard University Press, Cambridge (1987)
3. Bratman, M.E., Israel, D., Pollack, M.E.: Plans and resource-bounded practical reasoning. In: Computational Intelligence, vol. 4, pp. 349–355. Blackwell Publishing, Malden (1988)
4. Camarinha-Matos, L.M., Afsarmanesh, H.: A Comprehensive Modeling Framework for Collaborative Networked Organizations. Journal of Intelligent Manufacturing 18(5), 529–542 (2007)
5. Cerami, E.: Web Services Essentials Distributed Applications with XML-RPC, SOAP, UDDI & WSDL, 1st edn. O'Reilly & Associates, Inc., Sebastopol (2002)
6. Corchado, J.M., Bajo, J., Abraham, A.: GERAmI: Improving the delivery of health care. IEEE Intelligent Systems, Special Issue on Ambient Intelligence 23(2), 19–25 (2008)
7. Corchado, J.M., Bajo, J., De Paz, Y., Tapia, D.I.: Intelligent Environment for Monitoring Alzheimer Patients, Agent Technology for Health Care. In: Decision Support Systems. Eslevier, Amsterdam (in press, 2008)
8. Georgeff, M., Rao, A.: Rational software agents: from theory to practice. In: Jennings, N.R., Wooldridge, M.J. (eds.) Agent Technology: Foundations, Applications, and Markets. Springer, New York (1998)
9. Greenwood, D., Lyell, M., Mallya, A., Suguri, H.: The IEEE FIPA approach to integrating software agents and web services. In: Proceedings of the 6th international Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS 2007, Honolulu, Hawaii, pp. 1–7. ACM, New York (2007)
10. Jayaputera, G.T., Zaslavsky, A.B., Loke, S.W.: Enabling run-time composition and support for heterogeneous pervasive multi-agent systems. Journal of Systems and Software 80(12), 2039–2062 (2007)
11. Li, Y., Shen, W., Ghenniwa, H.: Agent-Based Web Services Framework and Development Environment. In: Computational Intelligence, vol. 20(4), pp. 678–692. Blackwell Publishing, Malden (2004)
12. Ossowski, S., Julián, V., Bajo, J., Billhardt, H., Botti, V., Corchado Rodríguez, J.M.: Open Issues in Open MAS: An abstract architecture proposal. In: 12th Conferencia de la Asociación Española para la Inteligencia Artificial (CAEPIA 2007), Salamanca, Spain, vol. 2, pp. 151–160 (2007)
13. Pokahr, A., Braubach, L., Lamersdorf, W.: Jadex: Implementing a BDI-Infrastructure for JADE Agents. In: EXP - in search of innovation (Special Issue on JADE), Department of Informatics, University of Hamburg, Germany, pp. 76–85 (2003)
14. Ricci, A., Buda, C., Zaghini, N.: An agent-oriented programming model for SOA & web services. In: 5th IEEE International Conference on Industrial Informatics (INDIN 2007), Vienna, Austria, pp. 1059–1064 (2007)
15. Schön, B., O'Hare, G.M.P., Duffy, B.R., Martin, A.N., Bradley, J.F.: Agent Assistance for 3D World Navigation. LNCS, vol. 1, pp. 499–499. Springer, Heidelberg (2005)

16. Shafiq, M.O., Ding, Y., Fensel, D.: Bridging Multi Agent Systems and Web Services: towards interoperability between Software Agents and Semantic Web Services. In: Proceedings of the 10th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2006), pp. 85–96. IEEE Computer Society, Washington (2006)
17. Snidaro, L., Foresti, G.L.: Knowledge representation for ambient security. In: Expert Systems, vol. 24(5), pp. 321–333. Blackwell Publishing, Malden (2007)
18. Tapia, D.I., Rodriguez, S., Bajo, J., Corchado, J.M.: FUSION@, A SOA-Based Multi-Agent Architecture. Advances in Soft Computing Series, vol. 50, pp. 99–107. Springer, Heidelberg (2008)
19. Tiwari, A.K., Shukla, K.K.: Implementation of generalized cross validation based image denoising in parallel virtual machine environment. Digital Signal Processing 14, 138–157 (2004)
20. Voos, H.: Agent-Based Distributed Resource Allocation in Technical Dynamic Systems. In: Proceedings of the IEEE Workshop on Distributed intelligent Systems: Collective intelligence and Its Applications (DIS 2006)., pp. 157–162. IEEE Computer Society, Washington (2006)
21. Wooldridge, M., Jennings, N.R.: Intelligent Agents: Theory and Practice. The Knowledge Engineering Review, vol. 10(2), pp. 115–152. Cambridge University Press, Cambridge (1995)