Towards a Methodology for Coordination Mechanism Selection in Open Systems

Simon Miles[1], Mike Joy[2], and Michael Luck[1]

[1] Department of Electronics and Computer Science
University of Southampton, Southampton SO17 1BJ, United Kingdom,
sm@ecs.soton.ac.uk / mml@ecs.soton.ac.uk
[2] Department of Computer Science, University of Warwick, Coventry CV4 7AL, United Kingdom, msj@dcs.warwick.ac.uk

Abstract. Agent-oriented software engineering (AOSE) is a promising approach to developing applications for dynamic open systems. If well developed, these applications can be *opportunistic*, taking advantage of services implemented by other developers at appropriate times. However, methodologies are needed to aid the development of systems that are both flexible enough to be opportunistic and tightly defined by the application requirements. In this paper, we investigate how developers can choose the coordination mechanisms of agents so that the agents will best fulfil application requirements in an open system.

1 Introduction

Connecting together software environments (subsystems) to form dynamic open systems offers applications the potential to make use of the most up-to-date functionality in any of those environments at any time. That functionality can then be extended by adding more environments and adding new services to the existing environments. Now, underpinning much of the *raison d'etre* for multi-agent systems is that they are a highly appropriate technology for applications that run in dynamic open systems. By reacting to changes in the system state and making appropriate decisions, agents can, if well designed, be *opportunistic*. Opportunistic agents attempt to take advantage of the best functionality available, regardless of whether those services have been implemented by the developer of the agents or provided by another. In order to ensure that an application can be judged to be opportunistic, the design should be tightly defined by the application requirements, i.e. the design should have *justification*.

How well an agent can take advantage of the functionality available in the system, and the amount of control it has over activities within the system depends on the *coordination mechanisms* it uses. A coordination mechanism is a software process employed by an agent to ensure other agents behave as it would find most useful, usually making use of communication between the agents. Different coordination mechanisms provide different functionality but also place different demands on the system. Following the work of others [4, 11], we suggest that the coordination model of a multi-agent system should be tailored to application requirements. We consider that an application may have a set of very different goals, i.e. provide a variety of functionality. Also, we consider that this functionality may be best achieved at any one time by agents under the

control of the application's designers or other accessible agents or processes within the open system.

We have developed a methodology in which we model cooperation to achieve an application's goals as the result of interactions between agents, without specifying which particular agents the interactions take place between. By raising *interactions* to the status of first-class design objects, we allow many agents, including those not existing at design time, to take on the role of participants in the interactions. The aim is that the most suitable agents within the open system can achieve the goals to the highest quality at any point in time.

With applications that have a set of differing goals, the agents that coordinate over those goals may appropriately use different coordination mechanisms. Choosing a single coordination model, such as a marketplace or a hierarchy, would not necessarily best fit the requirements. We therefore propose analysing each goal to determine the most suitable coordination mechanisms for use by agents cooperating to achieve the goal. We can then design agents that ensure application goals are achieved within the open system, based on the analysis results. Designing the multi-agent system for agent interactions where the participating agents are not known in advance places further demands on the analysis process.

In this paper, we demonstrate that our *assurance analysis* design process meets these demands, with a small example. In order, we present the following.

- A case study is provided to illustrate the methods described in this paper (Section 2).
- The case study requirements are analysed to derive the application *goals* and *preferences* as to how to best achieve the goals (Section 3.1).
- We discuss how a designer can choose from known agent coordination mechanisms to make sure the application is opportunistic and its design is justified (Section 3.2).
- Details of this approach are explained and illustrated by analysing the goals of the case study application (Section 3.3).
- After goals are matched to the coordination mechanisms which would best allow them to be achieved, we discuss how the designer can use this information to design the application agents. These agents will use the resources of the open system to achieve the application goals (Section 3.4).
- To show that the design is justified as we have claimed, we show how the design decisions made in the case study can be traced back to the requirements (Section 4).

Agent Interaction Analysis Our methodology, as a whole, is called *agent interaction analysis*. Using the methodology, a designer will perform broadly the following steps to create a multi-agent application functioning in an open system.

- Analyse the application requirements to discover the *goals* of the application and the *preferences* (or non-functional goals) that specify and/or quantify the priorities in achieving each goal (such as speed, accuracy of results etc.) Other information, such as how the goals are triggered in the system, e.g. by request from the user, and how goals can be decomposed into subgoals are also determined.

– The ideal target application design is then viewed in terms of agent interactions. The application can be seen as the achievement of the application goals within the open system due to cooperation between agents, each goal achieved being represented by an agent interaction. The agents taking part in the application are not specified as they cannot all be known about at design time ([8] also discusses designing systems that allow interactions between dynamically created agents).
– To achieve this target design, the designer adds agents to the open system to ensure the application goals are met.
– The designs of individual agents are tailored to match the preferences of the application, so as to best achieve the goals. The designer chooses between different models for each agent mechanism, where the models are described by third-party design patterns. The approach aims to preserve flexibility in the system as far as possible to allow full use of other agents in the open system that may more closely match preferences in their activity than the ones that the designer has added.

In this paper, we concentrate only on choosing the coordination mechanisms for the agents added to the system, which is a later stage of our methodology. As stated above, the coordination mechanisms most suitable for achieving a goal are those that tailor the interacting agents to best match the goal's preferences. See [10] for more on our methodology, and [13] for an overview of agent-oriented software engineering.

2 Case Study

To illustrate the design of multi-agent applications, we provide a case study that will be used throughout this paper. We assume that the case for using an agent-based approach for this application has already been made. In this section, we present the requirements document for an example application.

We require a collaborative weather mapping application. Using the application, a global weather map giving the current state is accessed and edited by various collaborating organisations. Contributors can add data they have gathered locally to the map in authorised locations. For example, one organisation may be authorised to add data concerned with a small local area, while another can add data for any location within a country. Authorisation is enforced to prevent accidental changes. Contributors and other organisations can access the weather data and be provided with predictions of weather at specified locations in the future. As with the contribution of data, different organisations have differing access rights to predictions on different locations.

Several services offer predictions based on the data, and the number of *predictors* available at any one time may vary, partly due to the load they each have on them. The predictor services vary in speed and in accuracy. They are represented as autonomous agents that accept prediction goals via published protocols. They must have access to the weather data to make the prediction, either by moving onto the system on which relevant data is stored or by repeated requests to the relevant sources. Prediction requests specify location and time (in the future), and the results should aim to be as accurate as possible. The application should be *opportunistic* in using open system resources, to ensure that operations are performed quickly or accurately, but should prioritise speed

over opportunism in general. Authorisation for editing data and viewing predictions is governed by stored *access rights*, which some users are authorised to edit.

Of course, requirements are unlikely to be provided completely in a single document, and further requirements capture may require further processes. Appropriate requirements capture and analysis is beyond the scope of this paper, and so we summarise key points that have been identified by the designer in our case study.

- Users need to perform three operations using the application as a whole: contribute some weather data, view predictions based on this data and set access rights for other users.
- When contributing data to the map, a user wishes the data to be added as quickly as possible, access rights to be observed and the data to be reliably integrated into the map, especially as new data may be generated frequently.
- When viewing a prediction based on existing data, a user particularly wishes for the quality of the prediction (the probability of its accuracy) to be high, as well as wishing it to be quickly presented to them. Again, the access rights should be observed, but also the user of the application wishes to ensure that it has enough flexibility to use higher quality prediction services when they become available.
- In setting access rights, the user primarily wishes the operation to be performed quickly and in observation of access rights.

3 Designing Coordination for an Open Application

In this section we examine how agents can be designed to fit application requirements. In particular, we examine which coordination mechanisms would best match the preferences of application goals, to allow an agent added to the system, or, indeed, already existing in the open system, to best fulfil the application goals opportunistically. The justification for each design decision is made explicit in sections marked 'Justification of Design Decision'.

3.1 Requirements to Design

Requirements engineering is the process of understanding and refining the requirements of an application. This may be aided by taking prospective users through usage scenarios [7], for example. Agent-oriented approaches to development should not force any change in an applications requirements. Therefore, the techniques used in requirements engineering for agent-based systems will be the same as those elsewhere [3, 12] (see Tropos [1] for an agent-oriented software engineering methodology based on requirements engineering).

A common requirements engineering technique is to describe the requirements in terms of functional *goals* it is intended to achieve and priorities and restrictions which, jointly, we call *preferences*. For brevity we exclude the requirements analysis of our case study and simply present the results in Tables 1 and 2 for goals and preferences respectively.

In Table 1, each goal identified is given a name, an end state and a list of associated preferences. The end state describes the state of the system in which the goal has been

Goal Name	Goal State Description	Preferences
Contributed Weather Data	A user has contributed weather data to the current map.	Security, Reliability, Speed
Viewed Prediction	A user has requested and received a prediction.	Security, Flexibiliy, Quality, Speed
Set Access Rights	A user has set the authorisation of another user for editing or viewing.	Security, Speed

Table 1. Goals identified by requirements analysis

achieved. The associated preferences state those priorities and restrictions that should be observed while attempting to achieve the goal. A goal is, in fact, a class of *goal instances*; there will be many times in which a user will contribute weather data, for example, and each of those may be the contribution of data at a different map location. Each contribution is an instance of the *Contributed Weather Data* goal. In Table 2, each preference identified is given a name and a description.

Preference Name	Preference Description
Security	Security of information prioritised.
Reliability	Reliability of achievement prioritised.
Flexibility	Flexibility for opportunistic behaviour prioritised.
Quality	Quality of results prioritised.
Speed	Speed of achievement prioritised.

Table 2. Preferences identified by requirements analysis

Justification of Design Decision The goals and preferences are decided upon by being directly identified from the requirements.

For a complex application, goals can be divided into subgoals. A subgoal will inherit the applicable preferences from its parent goal. This is not discussed further here but see [10] for more.

3.2 Coordination Mechanisms

An application that is opportunistic will use the services available in an open system to best achieve the application goals. The designer of an open system application will implement this by adding functionality to the system that *coordinates* the services to best effect. In an agent-oriented approach, the functionality added is viewed in the form of agents. Agents coordinate activity between themselves and other services by using *coordination mechanisms*. For example, here are brief descriptions of two such mechanisms.

Trust An agent using this mechanism, based broadly on the one suggested by Marsh [9], keeps a record of the success of cooperating agents in achieving goals. Future choices of cooperating agents are decided by whichever agents have been most successful in the past. This allows the agent to choose between agents based on their *trustworthiness* [5]. For Contributed Weather Data and Set Access Rights goals, the successfulness of an attempt could be the speed at which the goal is achieved. For Viewed Prediction, the success is judged by the prediction's accuracy, which could be checked when the actual data for the predicted time and location is contributed. The mechanism may take longer to use than others as the agent will have to discover which potential cooperating agents are available before choosing one. There is a wide range of research on modelling trust [5], and we do not claim that this mechanism as described is better or worse than others.

Forced Cooperation An alternative method for an agent to find suitable cooperating agents is for the designer to implement agents with references to pre-selected agents that are known to be tailored to achieve a goal and are forced to accept requests for cooperation over that goal. This mechanism, similar to standard method invocation in object-oriented systems, prioritises speed and reliability over opportunism.

Different goals have different measures of success, as given by the preferences such as those given in the preceding section. In order for a coordination mechanism to be suitable for matching a preference, a mechanism must have two properties. First, it must be useful in choosing agents or other services in the open system whose activity best matches the preference. Second, it must match the preference itself. For example, in our case study, contributing weather data should occur as quickly as possible. A mechanism coordinating this goal should choose agents offering to edit the appropriate part of the weather map and should not slow the process down unduly by the act of coordination.

In *agent interaction analysis*, the designer performs the following steps to decide on which coordination mechanisms is most appropriate for each application goal.

1. The designer is supplied with a set of coordination mechanism definitions in a standard pattern language. This follows the approach of *design patterns* [6], in which abstract parts of a design are made available to designers to encourage *re-use* of well-founded designs. There are many agent coordination mechanisms suggested in the literature but we use only the above two as examples in this paper (for brevity).
2. For each application goal, the designer chooses a coordination mechanism most suitable for matching the goal's preferences. In order to aid comparison, the designer performs more detailed analysis of the mechanisms. This analysis is called *assurance analysis* and is described in the next section.
3. From the choices of coordination mechanisms for each goal, the designer decides on the coordination mechanisms for the agents that will be added to the open system. When these agents are added the appication will be instantiated. This step is called *collation* and is discussed in Section 3.4.

In the full methodology, the above steps are performed for mechanisms other than coordination mechanisms, e.g. plan execution mechanisms, action scheduling mechanisms etc. In order to aid the comparison described in the first step, the coordination mechanisms are written in a pattern language. Examples for trust and forced cooperation are

shown in Tables 3 and 4. The tables describe eight aspects of each mechanism that may be revelant to application preferences.

Part Name	Coordination
Model Name	Trust
Description	The agent using this mechanism checks the quality of any solution provided by an agent and uses these assessments to decide which offers to accept in the future. The checks can be either by observation of the state achieved, if the goal attempts to achieve a particular observable state, or by an independent production of the same information, if the goal attempts to derive some information.
Algorithms	1. Send requests to agents; 2. Wait for a suitable duration to receive offers; 3. If no offers received, resend requests; 4. If some (one or more) offers are received, assess them to determine which comes from the most trustworthy agent; 5. Accept the offer from the most trustworthy agent.
Priorities	A trust-based mechanism prioritises quality of solution and reliability in obtaining a solution.
Resource Use	The agent using the mechanism will need to possess quantitative assessments of the trustworthiness of other agents, which rise and fall depending on observed quality of solution. The agent will make observations or request extra information for each goal.
Support Required	As the agent using the trust mechanism algorithm must wait a specified duration, it requires a scheduling mechanism capable of this.
Scaling	With a large number of possible collaborators for a goal, the number of models possessed by the agent will also be large. The observational checks will add to the time taken to process each goal by the agent.
Applicability	Where the quality of the goal is able to be checked in some way and is of more importance than speed or the low use of resources.
Problems	A trust-based mechanism may add a significant amount of processing to each goal the agent seeks cooperators for.

Table 3. A pattern for a coordination mechanism.

3.3 Assurance Analysis

The choice between mechanisms, even when design patterns are provided, may be difficult because different stages of the mechanisms' use can have very different effects on an application's performance. For example, with a trust mechanism such as the one suggested above, the process of building up a large table of trustwothiness information on agents in the open system may be costly in terms of time and storage space. It would, however, be very useful in making coordination decisions on which agents to best cooperate with. In order to supplement the information given in the supplied patterns, an application designer can further analyse how well mechanisms meet the application requirements by breaking their operation into smaller parts.

Assurance analysis is a procedure in which to analyse how well a coordination mechanism matches an application goal's preferences, which therefore aids comparison of coordination

Part Name	Coordination
Model Name	Forced Cooperation
Description	In this model, certain agents are required to cooperate over a goal on demand and are known to the agent employing this mechanism. The mechanism is approximately the same as message passing in (concurrent) objects.
Algorithms	To request cooperation from a forced agent, there is only one step. 1. Demand cooperation over the goal
Priorities	The model prioritises speed, reliability that the cooperators will be capable (through explicit design) and security in knowing the cooperator is pre-determined to be trustworthy.
Resource Use	Local information regarding the forced cooperation in agents is required by the agent employing this mechanism.
Support Required	The model requires mandatory adoption of goals in some agents providing the capability for this goal.
Scaling	Scales easily as it requires no communication beyond the minimum demand for cooperation, but does require suitable functionality to continue to be available within the application over time.
Applicability	This model is most applicable where security or reliability are of much greater importance than opportunism and where it is known that the forced functionality will always be available within the application.
Problems	Forced cooperation allows no flexibility in choosing cooperators so provides for no opportunism in exploiting the open system.

Table 4. A pattern for a coordination mechanism.

mechanisms for the application. It takes the viewpoint of a *single* agent attempting coordination with unknown others, so as to ensure flexibility in making full use of the open system. In this paper, we use simple tables to provide the results of the analysis for brevity.

In Tables 6, 7 and 8, we analyse and compare how well each of the two coordination mechanisms matches the preferences of each goal of our case study. The analysis attempts to be detailed to guide the designer in considering all aspects of the decision. Each cell in the tables is completed by the designer, and indicates whether a coordination mechanism is suitable for the goal being analysed, in some respect. Clearly two different designers may give different analyses, and so different entries into the tables, but the analysis still gives a *justification* for design decisions. Any more objective measure would be difficult as the priorities (*preferences*) of applications will vary widely.

Each of the tables is divided into four stages of execution. The top stage analyses the obtaining of information required to coordinate. The next stage down examines the updating of stored information on potential cooperating agents. The third stage concerns the analysis made by the agent from the cooperating agent information. The lowest stage examines acting on the analyses made.

In each table, a column represents analysis of a single mechanism while a row represents analysis of a single preference. For each cell in the table, the designer asks the following question: "Does coordination mechanism *X* allow preference *P* to be matched in stage *S*." In each cell, the designer answers 'yes' or 'no' to the question. In the lowest stage, the designer can see the overall suitability of the mechanisms. In general, the mechanism that answers 'yes' to all the questions is the best to choose though a compromise may be necessary if no mechanism matches every preference.

For example, in Table 6, the Trust mechanism has a 'no' entry in the second stage for the preference to prioritise speed. This is because, at the stage of building up information on potential cooperating agents, the Trust mechanism could be slow and prioritises quality of solution over speed. It can be seen that the analysis of a mechanism with regard to a preference is independent of any goal, so the same analysis of the Speed preference is made in Tables 6 and 8, for instance. In Table 7, the Forced Cooperation mechanism is stated not to allow the prioritisation of flexibility or quality of solution from the point at which it gathers information for cooperation (the first stage). This is because it does not attempt to discover any more suitable agents in the system but instead relies on pre-selected agents.

The analyses made suggest that the Forced Cooperation mechanism is most useful for the Contributed Weather Data and Set Access Rights goals and Trust is best for coordinating over Viewed Prediction.

Justification of Design Decision The analysis of how well each coordination mechanism matches the preferences of each goal justifies the choice of mechanisms.

3.4 Collation

Once designers have decided which of the coordination and other mechanisms available best match the application preferences, they can use this information to decide on the design of agents. The agents, when added to the open system, should instantiate the application goals while matching the preferences.

In our example, we have decided that a Forced Cooperation coordination mechanism would be best for coordinating Contributed Weather Data and Set Access goals and Trust would be most suitable for coordinating Viewed Prediction goal. As there are no other mechanisms specified in this brief example, the most obvious agents to introduce are to have one tailored to adopting and coordinating over the Contributed Weather Data and Set Access goals and one for the Viewed Prediction goal.

We give the final agent design in Table 5, which includes both the coordination mechanisms and the goals that each of the agents will make offers to achieve if requested (the goals they *will adopt*). The first agent can only adopt the Contributed Weather Data and Set Access Rights goals because it requires pre-selected cooperating agents and will only have them for those goals. We allow the second agent to adopt any of the goals, as there is no reason to restrict it and therefore, for maximum opportunism of the application, we do not restrict it (though it will be restricted in implementation). The full criteria for deciding on the complete set of agents being added to an open system depends on the application-wide preferences given in the requirements. Techniques for assessing whether applications meet application-wide preferences is discussed in [2].

The above application of *assurance analysis* is fairly simple as our case study is also simple and contrived for the purpose of this paper. For this reason we cannot illustrate the full potential of the analysis here. For example, both of the Forced Cooperation and Trust mechanisms concentrate on choosing between suitable agents which leads to preferences being disallowed at the early stages (higher tables in the analysis) or not at all.

Agent	Coordination	Will Adopt
1	Forced Cooperation	Contributed Weather Data, Set Access Rights
2	Trust	All goals

Table 5. Agents produced by collation

Information Acquisition

Preference	Trust	Forced Cooperation
Security	yes	yes
Reliability	yes	yes
Speed	yes	yes

Updating Models

Preference	Trust	Forced Cooperation
Security	yes	yes
Reliability	yes	yes
Speed	**no**	yes

Information Analysis

Preference	Trust	Forced Cooperation
Security	yes	yes
Reliability	yes	yes
Speed	**no**	yes

Acting on Analysis

Preference	Trust	Forced Cooperation
Security	yes	yes
Reliability	yes	yes
Speed	**no**	yes

Table 6. Assurance Analysis of Contributed Weather Data goal

Information Acquisition

Preference	Trust	Forced Cooperation
Security	yes	yes
Flexibility	yes	**no**
Quality	yes	**no**
Speed	**no**	yes

Updating Models

Preference	Trust	Forced Cooperation
Security	yes	yes
Flexibility	yes	**no**
Quality	yes	**no**
Speed	**no**	yes

Information Analysis

Preference	Trust	Forced Cooperation
Security	yes	yes
Flexibility	yes	**no**
Quality	yes	**no**
Speed	**no**	yes

Acting on Analysis

Preference	Trust	Forced Cooperation
Security	yes	yes
Flexibility	yes	**no**
Quality	yes	**no**
Speed	**no**	yes

Table 7. Assurance Analysis of Viewed Prediction goal

Information Acquisition

Preference	Trust	Forced Cooperation
Security	yes	yes
Speed	**no**	yes

Updating Models

Preference	Trust	Forced Cooperation
Security	yes	yes
Speed	**no**	yes

Information Analysis

Preference	Trust	Forced Cooperation
Security	yes	yes
Speed	**no**	yes

Acting on Analysis

Preference	Trust	Forced Cooperation
Security	yes	yes
Speed	**no**	yes

Table 8. Assurance Analysis of Set Access Rights goal

Justification of Design Decision The design of agents produced by collation is wholly based on the mechanisms determined to be best for each goal.

4 Judging the Design

When creating a multi-agent application, the design is justified by the requirements if the functionality is divided between agents in a way justified by the requirements. The dynamic nature of open systems means that any test performed on an implemented (opportunistic) open system application is unrepeatable and, therefore, of limited value [14]. Also, the application may make use of any functionality in the open system but, clearly, the only part of the application that can be tailored to the requirements is that which is known about at design time and whose form is under the control of the designer.

4.1 Tracing Backwards

The first agent in Table 5 has the functionality identified as most suitable for originators of two goals: *Contributed Weather Data* and *Set Access Rights*. The designer has chosen to merge the agents designed to coordinate over these goals, so that the organisation contains one agent (in the application set) tailored to providing this functionality rather than two. By simply identifying agents in the original requirements, the designer could, for example, have chosen to implement two agents that separately dealt with the two goals. To see why the decision to choose this particular organisation is justified, we can reason (trace) backwards from the *collation* stage, at which the final organisation is chosen.

1. At the start of the collation stage, the designer has decided that an agent tailored to coordinating over the Contributed Weather Data goal would have an architecture which used forced cooperation to coordinate. The designer has also decided that an agent tailored to coordinating over the Set Access Rights goal would have the same architecture. These are decisions on the most suitable architectures for agents initiating cooperation over the goals. *The decisions on the choice of most suitable architecture for each agent is the results of analysis, such as the* assurance analysis *described earlier, into how the agents can be tailored to match the requirements, and should, therefore, inform the choice of organisation.*
2. In collation, the designer decides which agents make up the final organisation based on the coordination mechanism design decisions for the agents. It would be likely that at least one agent with the Forced Cooperation mechanism for coordinating over the Contributed Weather Data goal is implemented so that when an agent, or the user, wishes to achieve an instance of the Contributed Weather Data goal, there will be an agent tailored to doing so in the application. The same is true for the Set Access Rights goal. However having two agents with the same architecture is not necessarily the best organisational division, as having two agents will use up more resources. *Separation of agents into more than one agent type (architecture) in the organisation is not justified by the requirements if the architectures of the*

agents are similar enough, as long as the architectures are themselves justified by the requirements.

3. To see that the architectures of the agents are justified we examine the *assurance analysis*. The assurance analysis for the Contributed Weather Data goal gives reasons why the most suitable coordination mechanisms for the goal is Forced Cooperation, based on comparing preferences of that goal with the operations of each mechanism. The Set Access Rights analysis comes to the same conclusion for that goal. *The choice of architectures for the agents was decided on the basis of the goal and application preferences*.
4. The Contributed Weather Data and Set Access Rights goals and their preferences were extracted from the requirements in the requirements analysis stage by examining the scenarios, entities and goals mentioned in the requirements.

4.2 Opportunism

The other aspect of justification for designs is that it is restricted as little as possible in its opportunism. The two agents in Table 5 differ greatly in their interoperability. The first agent is heavily restricted in its operations while the second is highly interoperable. We claim that the opportunism of each is restricted only as far as the requirements demand and the design decisions are, therefore, justified. We examine the design decisions restricting or allowing opportunism for each agent below.

1. The first agent is heavily restricted in its activity.
   (a) The preference to validate authorisation to edit the weather map and access rights and the emphasis on reliability in contributing to the map require that the agent uses only trusted agents in making alterations. This can either be achieved by checking the agent's actions to determine their reliability or by always using standard cooperating agents chosen by the designer. Speed is more important than interoperability so the designer chose the latter option (the Forced Cooperation coordination mechanism).
   (b) Using Forced Cooperation requires that references to cooperating agents be known in advance. Thus, only the goals for which references are provided can be attempted by the agent, which are Contributed Weather Data and Set Access Rights in this case. The agent will only offer to adopt these two goals.
   (c) These restrictions are derived from the requirements (goals and preferences).
2. The second agent has very little restriction on its behaviour.
   (a) The preferences on getting accurate predictions in prediction viewing operations lead the designer to use interoperation to find the most suitable cooperating agents at each instance. To decide between cooperating agents, the agent uses a trust-based mechanism giving information on the previous likely accuracy of results.
   (b) There is no reason to prevent the agent from being given the ability to coordinate over other goals, and so it is given the ability to do so.
   (c) The requirements encourage opportunism in this case, and the design of the agent reflects this.

5 Conclusions

In this paper, we have shown how a designer can choose coordination mechanisms for agents implementing an application in an open system. We have shown that these decisions can be analysed to determine that the design does meet the requirements, and allows for flexible use of the functionality available in the open system at any one time.

Our methodology was developed to allow designers to produce opportunistic open systems applications whose design is fully justified by the application requirements. We do not believe this is currently achieved by other agent-oriented software engineering methodologies.

Future work will examine how the infrastructure supporting agents can also be designed to cohere with the guiding concepts identified above. One minor problem to be addressed with the methodology, however, as with any that attempts to analyse all points at which an application could take advantage of available functionality, is keeping the volume of the specification to a manageable level. More detailed results of the case study, and further details of the methodology, are available from the first author.

Acknowledgements This work was carried out while supported by the myGrid project (EPSRC reference GR/R67743/01) at University of Southampton.

References

1. J. Castro, M. Kolp, and J. Mylopoulos. A requirements-driven development methodology. In *Proceedings of the 13th International Conference on Advanced Information Systems Engineering (CAiSE-01)*, Interlaken, Switzerland, 2001.
2. P. Davidsson and S. Johansson. Evaluating multi-agent system architectures: A case study concerning dynamic resource allocation. In *this volume*, Madrid, Spain, 2002.
3. A. M. Davis. *Software Requirements: Objects, States and Functions*. Prentice Hall, 1993.
4. V. Dignum, H. Weigand, and L. Xu. Agent Societies: Toward Frameworks-Based Design. In M. Wooldridge, P. Ciancarini, and G. Weiss, editors, *Proceedings of the Second International Workshop on Agent-Oriented Software Engineering (AOSE-2001)*, pages 25–32, Montreal, Canada, 2001.
5. R. Falcone and B. S. Firozabadi. The challenge of trust: The Autonomous Agents '98 Workshop on Deception, Fraud and Trust in Agent Societies. *Knowledge Engineering Review*, 14(1):81–89, 1999.
6. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design patterns: Abstraction and reuse of object oriented design. In *Proceedings of ECOOP'93, Kaiserslautern, Germany*, 1993.
7. P. Haumer, K. Pohl, and K. Weidenhaupt. Requirements elicitation and validation with real world scenes. *IEEE Transactions on Software Engineering*, 24(12), December 1998.
8. M. N. Huhns. Interaction-oriented programming. In P. Ciancarini and M. J. Wooldridge, editors, *Proceedings of Agent-Oriented Software Engineering 2000 (AOSE 2000)*, 2000.
9. S. P. Marsh. *Formalising Trust as a Computational Concept*. PhD thesis, Department of Computing Science and Mathematics, University of Stirling, 1994.
10. S. Miles, M. Joy, and M. Luck. Designing agent-oriented systems by analysing agent interactions. In P. Ciancarini and M. J. Wooldridge, editors, *Proceedings of Agent-Oriented Software Engineering 2000 (AOSE 2000)*, 2000.

11. A. Omicini. SODA: Societies and infastructures in the analysis and design of agent-based systems. In P. Ciancarini and M. J. Wooldridge, editors, *Proceedings of Agent-Oriented Software Engineering 2000 (AOSE 2000)*, 2000.
12. A. van Lamsweerde. Requirements engineering in the year 00: A research perspective. In *Proceeding of the Twenty-Second International Conference on Software Engineering (ICSE-00)*, 2000.
13. M. Wooldridge and P. Ciancarini. Agent-oriented software engineering: The state of the art. In P. Ciancarini and M. J. Wooldridge, editors, *Proceedings of Agent-Oriented Software Engineering 2000 (AOSE 2000)*, 2000.
14. F. Zambonelli and H. V. D. Parunak. Signs of a revolution in computer science and software engineering. In *this volume*, Madrid, Spain, 2002.