Designing Institutional Multi-Agent Systems*

Carles Sierra[1], John Thangarajah[2], Lin Padgham[2], and Michael Winikoff[2]

[1] Artificial Intelligence Research Institute (IIIA)
Spanish Research Council (CSIC)
Catalonia, Spain
sierra@iiia.csic.es
[2] School of Computer Science and Information Technology,
RMIT University,
GPO Box 2476V, Melbourne, VIC 3001, Australia
{johthan, linpa, winikoff}@cs.rmit.edu.au

Abstract. The vision of agents working together on the Internet, in virtual organizations, is one that is increasingly common. However, one of the issues is the regulation of the participating agents and their behaviour. A substantial body of work exists that investigates agent societies and agent organizations, including work on *electronic institutions*, such as *Islander* and *Ameli*. However, although such work provides concrete tools for specifying and enacting institutions, there is a lack of clear documented guidance to designers who are using these tools. In this paper we describe a methodology for developing an institutional structure for multi agent systems. This methodology captures the knowledge and experience within the Islander group, and integrates it with the *Prometheus* methodology.

1 Introduction

The vision of agents working together on the Internet, in virtual organizations, is one that is increasingly common. One of the issues however is the regulation of the participating agents and their behaviour. There is a substantial body of work that investigates agent societies and agent organizations (for a review see [1]) and electronic institutions (e.g. [2–4]). Methodologies for designing agent organizations such as OperA [5] have been used for developing industrial systems, and there are various examples of implemented systems. However there is very limited support for developing such systems, in terms of runtime platforms or design tools. One exception is the Islander design tool [6] and the Ameli runtime platform [7]. Together these provide an environment for designing and developing electronic institutions. Currently there is no written methodology or guidance on how to actually use these tools to develop electronic institutions, although there is considerable experience developed within the Islander group. This paper provides a methodology for developing an institutional structure for multi agent systems, capturing the knowledge and experience within the Islander group into a *Social Design*

* This work was supported by the Australian Research Council under grant LP0453486, in collaboration with Agent Oriented Software. We also thank the Australian Tourism Data Warehouse for use of their tourism content in our agents. Carles Sierra is being supported by the Spanish Web-I(2) project and the ARC Discovery Grant DP0557168.

phase, within a slightly modified version of the Prometheus methodology [8], a practical agent-oriented software engineering methodology that aims to be usable by software developers and undergraduate students.

Design tools play an extremely important part in supporting the development of complex systems. Consequently this Social Design phase has been developed specifically to work with the Islander design tool. As the area of methodological support for design and development of agent organizations, institutions and societies matures, it is likely that this would be generalized to provide an approach less dependent on the particular available toolset. However at this stage, we believe it is useful to provide a very concrete methodology that gives sufficient guidance to the developer that they can successfully develop a system.

The approach that we have taken is to use the Prometheus Design Tool (PDT[3]) [9] and a variant of the Prometheus methodology for doing the initial analysis and system specification. The system specified may include the Electronic Institution as a part of the system, or it may be the entire system. Using this initial design in PDT, the design of the Electronic Institution component is then carried out within the Islander tool, and the outcome of this provides information back into the Prometheus design process for those parts of the system that lie outside the actual Electronic Institution infrastructure.

In the rest of this paper we provide some background on Islander, and the view of Electronic Institutions that Islander and Ameli are designed to support. We then describe in detail the design process for developing an Electronic Institution, embedding this within the Prometheus methodology. In order to illustrate the design process in a concrete way, we take an example of an (very limited) Electronic Institution for travel bookings, and use this for illustration throughout the methodological description.

2 Background: ISLANDER

The idea behind Electronic Institutions (EIs) is to mirror the roles traditional institutions play in the establishment of "the rules of the game"– that is, the set of conventions that articulate agents' interactions. The essential roles EIs play are both descriptive and prescriptive: the institution makes the conventions explicit to participants, and it warrants their compliance. Development environments to realize Electronic Institutions involve a conceptual framework to describe agent interactions as well as an engineering framework to specify and deploy actual interaction environments. Work on such a development environment has been happening for some time within the Intelligent Agents Group at the Artificial Intelligence Research Institute of Spain [2–4, 10]. Considerable experience in the deployment of applications as EIs (e.g. [11, 12]) provide confidence in the validity of the approach.

EIs are socially-centered, and neutral with respect to the internals of the participating agents. They provide a regulated virtual environment where the relevant interactions among participating entities take place. The Electronic Institution provides an infrastructure which ensures, as well as specifies, legitimate interactions. In order to realize this infrastructure all interactions are considered to be speech acts, and any effect on

[3] Freely available from www.cs.rmit.edu.au/agents/pdt

the shared environment is considered to happen only as a result of illocutions uttered by participating agents.

The Islander tool supports specification of an EI which is then executable using the Ameli runtime environment[4]. Conceptually there are four main areas to be specified using Islander, and we will describe each in turn. These are:

- **The Dialogical Framework** which specifies the roles within the particular domain and the ontology.
- **The Interaction Structure** which describes the scenes, the pattern of allowable interactions within each scene, and also the effect these interactions have within the shared environment.
- **The Performative Structure** which provides an overview of the connections between different scenes and possibly other (sub-)Performative structures, and the role-flow policies.
- **Norms and Constraints** which capture rules which will be enforced by the EI.

2.1 Dialogical Framework

A *role* defines a particular pattern of behaviour and all participants within an EI take on a particular role. For example, in an auction house there may be buyers and sellers. Participants may change their roles over time, for example an agent acting as a buyer at one point may act as a seller at another. It may also be the case that we restrict an agent from acting as a buyer and seller at the same time, this is done by specifying a particular relationship between roles called DSD (Dynamic Separation of Duties). A stronger version of this relationship, called SSD (Static Separation of Duties) prevents agents from playing two incompatible roles within an institution even if they are played at different times.

We also need to distinguish between internal and external roles. The internal roles define a set of roles that will be played by *staff* agents which correspond to employees in traditional institutions. Agents that are external to the institution cannot take on these roles and are restricted to external roles. When defining an EI we need to consider the roles that participants may take on, whether the roles are internal or external and the relationship between roles if any.

We need to settle on a common illocutory language that serves to tag all pertinent interactions, or more properly, the valid speech acts. Formally, we consider each interaction to be an illocutory formulae: $\iota(speaker, listener, \varphi, t)$. The speech acts that we use start with an illocutory particle (inform, request, accept, ...) that a $speaker$ addresses to a $listener$, at time t, and the content φ of the illocution is expressed in some object language whose vocabulary is the EI's *ontology*. The $speaker$ and $listener$ are roles within the EI.

To fill in these formulae therefore, we need vocabulary and grammar, and we need to refer to speakers and listeners, actions and time. We call all this the *Dialogical Framework* because it includes all that is needed for agents to participate in admissible dialogues in a given EI. Two important aspects of the Dialogical Framework are the *So-*

[4] Further details about electronic institutions can be found at http://e-institutor.iiia.csic.es.

cial Structure model which captures the roles and their relationships, and the *Ontology model* which defines the entities in the domain.

2.2 Interaction Structure

Interactions between agents are articulated through recurrent dialogues which we call *scenes*. Each scene follows some type of conversation protocol, that restricts the possible interactions between roles. Scenes also represent the context in which the uttered illocutions must be interpreted, as the same message may have different meanings in different contexts.

The protocol of a scene is specified by a directed graph whose nodes represent the different states of a dialogical interaction between roles (e.g. see figure 6). Each state indicates the agents that are allowed to enter or leave a particular scene. The transitions from one state to another are labeled with illocution schemata from the scene's dialogical framework (whose sender, receiver and content may contain variables) or timeouts. These transitions may also have constraints and actions attached. Constraints are used to restrict the paths that the scene execution can follow. For example, in an auction scene it is possible to specify as a constraint, that a buyer can only submit a bid that is greater than the previous bid. Actions are used to specify any updates to the shared state of the institution when a transition occurs.

At execution time agents interact by uttering grounded illocutions matching the specified illocution schemata, and so binding their variables to values, building up the *scene context*.

2.3 Performative Structure

Activities in an electronic institution are organized in a *performative structure* as the composition of multiple, distinct, and possibly concurrent, dialogical activities, each one involving different groups of agents playing different roles.

A performative structure can be seen as a network of scenes, whose connections are mediated by transitions. It determines the role-flow policy among the different scenes by showing *how* agents, depending on their roles, may move into different scenes (other conversations), and showing *when* new scenes (conversations) are created.

In all EIs we assume that there is always an initial and a final scene, which are the entry and exit points of the institution. Each scene can have multiple instances at runtime. An example is shown in figure 5 on page 13. Rounded rectangles depict scenes, and arcs between them indicate the paths that agents can take.

A Transition can be thought of as a gateway between scenes or as a change of conversation. When an agent leaves a particular scene, there are different transitions that could happen: An *Or* transition allows an agent to choose which target scene(s) to enter. An *And* transition forces agents to synchronize before progressing to different scenes together. The arc of a transition to a target scene is labeled with *new* if the scene is created for the first time, and *one*, *some* or *all* indicating if the agent enters one, some or all instances of the target scene type.

In each transition it is possible to also specify if an agent takes on a different role when entering a new scene. Hence the performative structure provides an overview of the path and roles taken up by an agent from the start to the end scene within an EI.

2.4 Norms and Commitments

The main purpose of an EI is to control the interactions between the participants and ensure that they all adhere to agreed rules.

Actions within an institution are speech acts. These speech acts create obligations or socially binding commitments whose fulfillment is ensured by the institution. We make such intended effects of commitments explicit through *normative rules*.

We define the predicate $uttered(s, w, i)$ that allow us to express the connection between illocutions and norms. It denotes that a grounded illocution unifying with the illocution scheme i has been uttered at state w of scene s (the state w is an optional element).

A normative rule is specified using the following three elements: (i) Antecedent: the actions that trigger the activation of the norm, expressed as a predicate defined above; (ii) Defeasible antecedent: the utterance which releases the agent from the obligations described in the consequent; also expressed as an $uttered$ predicate; and (iii) Consequent: the set of obligations. An obligation is expressed as $Obl(x, i, s)$, denoting that the agent bound to role x is obliged to utter i in scene s.

3 Designing Institutional MAS with Islander and Prometheus

In exploring how the design of an Islander EI is typically done, we have identified that it is useful to begin with a slight modification of the Prometheus System Specification phase.

This phase in Prometheus consists of a number of interleaving, iterative steps, to define goals, roles, scenarios and environmental interface in the form of actors, actions and percepts. These elements, as well as the *ontology*, then provide input to a new phase which we call the *Social Design* phase, where the EI can be specified in the Islander tool. Specifically, the roles are incorporated into Islander's social structure, the scenarios are used as a starting point for developing the performative structure and the interaction model, goals are used to help identify norms, and, of course, ontology elements identified in the system specification phase feed into developing the ontology in the Social Design phase (see figure 1).

Once the Electronic Institution has been designed, there is additional information which can be provided back into Prometheus' Architectural Design phase, in order to design the agents that will join the institution. The institution is used as a starting point for the design. Specifically, the social structure and the norms identified in the Social Design phase are used, in addition to the system interface and goals, in determining which agent types should exist; and the interaction model is used as a starting point for defining interaction protocols.

In order to illustrate the methodology in greater detail we use a simple example of an EI for making flight and hotel arrangements. In this EI, agents can make and accept

flight and hotel bookings, including payment for these. The idea is that the EI provides a trusted interaction space in which agents can engage in these interactions with other agents.

The remainder of this section explains the design methodology in greater detail, using this example. Section 3.1 covers the System Specification phase, with particular focus on the changes that have been made to the existing phase in Prometheus. Section 3.2 covers in detail the (new) Social Design phase, and section 3.3 briefly indicates how the Architectural Design phase has been modified to make use of the information produced in the Social Design phase. The detailed design phase is unchanged and is not discussed in this paper.

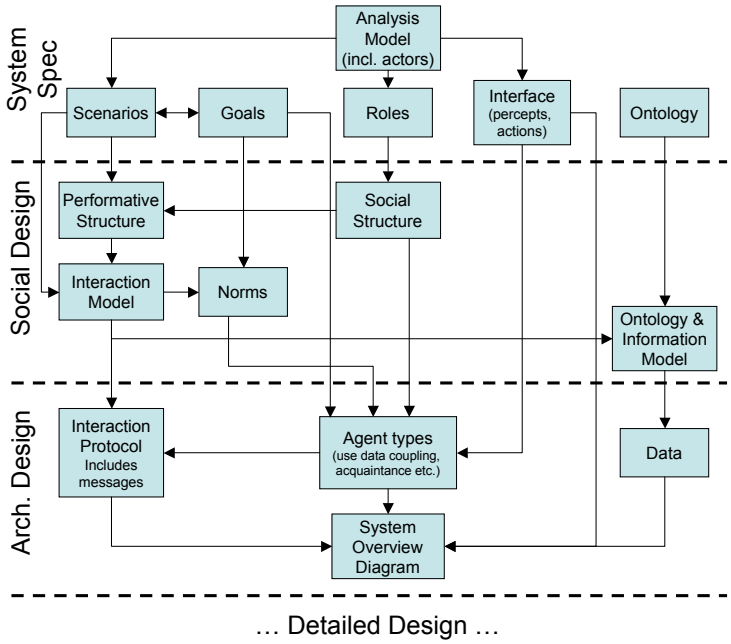

Fig. 1. Revised Methodology

3.1 System Specification

In order to appropriately use the Prometheus System Specification phase in conjunction with designing an electronic institution, there are a few changes that need to be made. In the standard Prometheus approach, we would start with identifying the actors, persons or entities, including software, that are external to the system, and that would interact with the system. In this case, the natural starting point is identification of the roles that will interact within the Electronic Institution. In our example these could be a

Customer[5] role, a Travel Agent role, and a Banker role. These are external to the Institution, so one option could be to model them as Prometheus actors. However we choose to retain actors as the entities external to both the Institution and the software agents which we are designing. In our example, the actors may be Airline companies (with whom our electronic travel agent must eventually make a booking), Hotel proprietors, and Human customers.

Having identified the roles, (which will eventually be played by agents) we then identify the main scenarios for the EI. In this case we identify a Travel Booking scenario and a Payment scenario. The Customer and the Travel Agent roles are involved in the Travel Booking scenario, while the Bank and the Customer roles are involved in the payment scenario (called Pay Booking). The details of the interaction of the roles, with respect to the scenario is left until the Social Design phase.

If desired, we can identify the external actors that our roles will interact with, and the percepts or actions that we expect to be a part of those interactions. This information can then be used in the design of the agents that will be able to fill these roles. We identify that the Travel Agent role can be expected to have a booking action for interaction with the Airline and the Hotel proprietor. (We may later decide that we also want a confirmation percept from actor to Travel Agent). Similarly we identify a Request percept from the human customer to the agent that will play the Customer role, and an action to Provide Itinerary (again, further interaction may well be defined later).

We also add to the System Design phase a step to identify *soft goals* and to link these to particular entities if appropriate. In our example we identify three soft goals: reliable service provision; safe transactions; and ability to hold reservations for three days. The first of these we leave unattached, while safe transactions is attached to the Pay Booking scenario, and the ability to hold (unpaid) reservations is attached to the Travel Booking scenario.

The resulting analysis overview diagram is shown in figure 2. Boxes with stick figures denote either roles or actors (the distinction is indicated in the name), large squares denote soft goals, and arrows-like icons (with the indication "scenario") denote scenarios. Percepts are star-bursts, and actions are arrows (e.g. Provide Itinerary).

Having identified the roles and the main scenarios, we then further develop both the goals and the scenarios.

Goals: Whereas in "classical" Prometheus, goals are all system goals, which will be eventually allocated to specific agents, here we distinguish between *three* types of goals: *individual* goals that are allocated to a role (and later to an agent type), *joint* goals that are achieved by a group of roles (eventually agents), and *social* goals, where the EI plays a part in ensuring that these goals are achieved.

In deciding what type a given goal should be we consider whether it belongs to a single role (so is probably an individual goal), or to multiple roles (in which case it is probably joint or social). If any of the roles that the goal is assigned to are institutional ("staff") roles, then it must by definition be a social goal. However, at this stage in the

[5] We use san serif font to indicate names in the design. Due to limitations with the tool multi-word names do not have spaces in the figures (e.g. TravelAgent), but, for readability, do have spaces in the paper's text (e.g. Travel Agent).

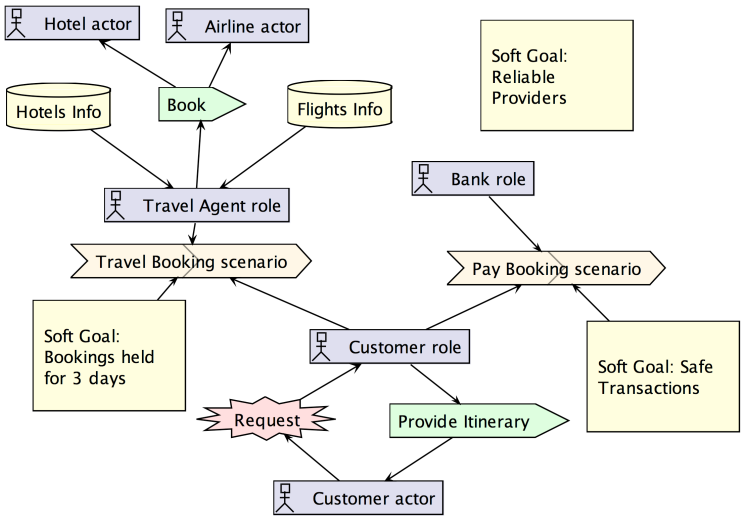

Fig. 2. Analysis diagram

process we may not have enough information to determine whether a goal should be joint or social, and so we may defer this decision until the Social Design phase.

As in "classical" Prometheus, the identification of system goals goes hand in hand with identification of scenarios and scenario steps. Goals are refined by techniques such as asking 'how can we achieve this goal?' [13]; and refinement and abstraction, along with combining similar subgoals that arise in different parts of the system, eventually leads to a well developed goal hierarchy.

In addition to the individual/joint/social distinction, we have also introduced *soft goals* and the Goal Overview Diagram has been extended to show these. Soft goals are (optionally) linked to goals and provide information on the desired properties of the system. This information is captured explicitly so that it can be used later to develop constraints (in scenes) or norms (see section 3.2).

For example, figure 3 shows the goal overview diagram for the travel agent example. It shows that the high-level goal Travel Booking is a joint goal, as are its three child goals (Find Flights, Find Hotels and Make Booking). However, the sub-goals of Pay Booking are clearly individual goals.

Scenarios: Prometheus scenarios, which are identified for each actor that will interact with the system, contain steps (which can be *goals*, *percepts actions* or *sub-scenarios*). In modelling scenarios within EIs it is natural to think of these scenario steps conceptually as *joint activities* involving some number of agents. Joint activities in our example would be Make Booking, Pay Booking, etc. The details that need to be recorded for a joint activity (name, roles, goal, and relevant data) are the same as for a scenario but without any steps. Consequently, rather than introduce joint activity as a new step type, we simply allow the steps of a scenario to be optional. A minor change is that we allow

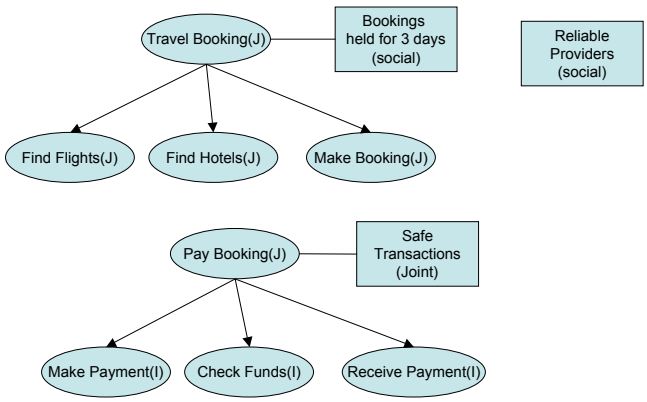

Fig. 3. Goal Overview Diagram. Ovals denote goals, and rectangles denote soft goals.

Scenario: Travel Booking
Goal(s): Travel Booking
Steps:

#	Type	Name	Role(s)	Data used	Data produced
1	Scenario	Find Flights	Flight Provider Customer	Flights Info	
2	Scenario	Find Hotels	Hotels Provider Customer	Hotels Info	
3	Scenario	Make Booking	Flight Provider Customer	Flights Info	Bookings
4	Scenario	Make Booking	Hotels Provider Customer	Hotels Info	Bookings
5	Scenario	Pay Booking	Bank, Customer	Bookings	Bookings, Flights Info
6	Scenario	Pay Booking	Bank, Customer	Bookings	Bookings, Hotels Info

Fig. 4. Example Scenario

the goal in the scenario descriptor to be a set of goals, interpreted as a conjunction, rather than a single goal. Figure 4 depicts the Travel Booking scenario with joint activities (scenarios) as the steps[6].

Roles and Ontology: In standard Prometheus roles are identified by grouping goals (along with percepts and actions) into clusters and identifying a role that would manage these goals. Here we have already identified some roles in the analysis overview diagram. The clustering of goals may identify additional roles. In our example we identify the additional roles of Flight Provider, Hotel Provider, and payer. We then extend

[6] Note that this scenario uses the roles of Hotel Provider and Flight Provider instead of the more general super-role Travel Agent.

the standard Prometheus process to consider and capture sub-role relationships. In our example we identify Flight Provider and Hotel Provider as sub-types of Travel Agent. The exclusion relationships between roles (static separation of duties (SSD), and dynamic separation of duties (DSD)), required by Islander, is however left until the Social Design phase.

In addition to the roles played by agents entering the Electronic Institution, Islander has a concept of *internal roles* which are played by staff agents, and are part of managing the infrastructure of the EI. We do not necessarily identify these internal roles during the System Specification phase, as they are typically introduced during the Social Design phase, where consideration is given to managing the infrastructure functions of the EI. If some such roles are identified at this stage, they should be marked as internal.

During scenario specification there is identification of data used and produced, which is the start of ontology definition, and is defined as such. In our example we identified Flight Info, Hotel Info and Booking as three necessary items in the ontology, and made some initial decisions about fields required.

Summary of modifications to Prometheus: There are five modifications to the standard Prometheus process in order to have a process which facilitates and feeds into the Social Design phase done in Islander. These are:

(a) The analysis overview diagram was changed to capture roles (that would be taken on by agents entering the EI) as well as the actors external to the system, and also to include soft goals.
(b) A role hierarchy is developed, if appropriate. Also a distinction is introduced between internal roles ("staff" roles of the EI) and external roles (to be taken on by agents operating within the EI).
(c) Steps are made optional in a scenario to allow use of scenarios for modelling joint activities.
(d) The goal overview diagram is extended to allow soft-goals to be captured, and to allow different types of goals to be distinguished.
(e) Identification of data in scenario steps is extended to a preliminary ontology development activity.

3.2 Social Design

The social design phase, specifying the details of the Electronic Institution (completed using the Islander tool), takes input from the Prometheus based system specification. We structure this phase as eight separate steps. Note that, as with all system design and development, these are iterative rather than strictly sequential. The steps, along with the part of Islander that is addressed in each step, are as follows:

1. Develop the social structure (roles and relationships)
   Social Structure model (in the Dialogical Framework)
2. List scenes with participating roles (input to step 3)
3. Develop the performative structure (network of scenes) and initial flow
   Performative structure model

4. For each scene, define the interaction structure: basic conversation stages, and flow of conversation
   Interaction Structure
5. Develop the ontology (influenced by interaction model)
   Ontology Model (in the Dialogical Framework)
6. Define the information model, actions and constraints
   Interaction Model
7. Identify and specify norms
   Norms and Commitments Model
8. Check that all social goals have been achieved

In particular, steps 4-6 are performed for each scenario and are very iterative. We have presented them as distinct steps for two reasons. Firstly, because the steps are concerned with different parts of Islander (e.g. steps 4 and 5 are concerned with the interaction model and the ontology model respectively). Secondly, because the sequencing of the steps can vary: one possibility is to perform steps 4-6 for one scene, then perform them for the next scene, and so on; but it is also possible to perform steps 4 and 5 for each scene in sequence, and only then continue with step 6.

The rest of this section describes these steps in detail, with reference to our travel example.

Step 1: Develop the social structure (roles and relationships). In this (simple) step we refine the roles identified in the system specification phase by adding further relationship information.

We begin by simply transcribing the roles that have been identified in the previous phase. In our example these are Travel Agent, Customer, Bank, Flight Provider and Hotel Provider. We then add any additional roles we recognize as being necessary internal roles (though these may well be added later when considering norms), and further develop the role structure if desired. In our example we decide to introduce an internal Reliability Monitor role to maintain information about providers in order to support the soft goal of "Reliable Providers".

Finally, we consider and specify exclusivity relationships: which roles cannot be filled by the same agent. As discussed in section 2, Islander defines both a static and a dynamic separation of duties. In our example, it is fairly clear that the Reliability Monitor should be separate from the provider or consumer of the service that is being monitored, and so we add an SSD (Static) relationship between the Reliability Monitor and the Travel Agent, and between the Reliability Monitor role and the Customer role.

Step 2: List scenes with participating roles. Having refined the roles that exist in the institution, the next step is to define the scenes that these roles will participate in. A good starting point for identifying scenes is to take the joint activities (sub-scenarios) that are the steps of scenarios in the previous phase. (These are primarily scenarios that have no sub-scenario steps.) It is also useful to consider whether certain scenarios can be generalized into a common scenario type which permits two or more of the existing scenarios to be merged. There are certain commonly-used types, such as information seeking, that can often be used to do this generalization.

For example, looking at the scenario in figure 4, we have six sub-scenarios that could become scenes. In this case we decide that finding a flight and finding a hotel may have significant differences in the information that is exchanged, but that once information has been found, booking a hotel and booking a flight are likely to be similar enough that they can be merged into a more generic booking scene. Similarly, paying for a flight and paying for a hotel are merged into a payment scene. This gives us the following scenes: Hotel Info, Flight Info, Booking, and Payment. Additionally, Islander requires a starting and ending scene (respectively called Enter and Exit), and so these are added.

If the scenario structure is deeply nested, then it may be useful to use nested performative structures as a way of modeling the interaction in such a way that the complexity at each level is manageable.

When defining scenes, we need to think about a number of properties of scenes such as cardinalities (will there be one instance of the scene or many?), what triggers scene creation, and, where there are multiple scene instances, whether agents join all scene instances, one instance only, or some subset of the scene instances. For example, for the Hotel Info scene we choose to have one scene per Hotel Provider, with the Customer choosing to join some subset of the available scenes. Since there is one scene instance per Hotel Provider, it makes sense for new scene instances to be created when a Hotel Provider moves into the scene.

Finally, we need to consider whether multiple scenes may map to the same underlying scene type. In Islander, nodes in the performative structure are scenes, which are instances of scene types. Although often there is a one-to-one mapping between scenes and scene types, in some cases, multiple scenes map to the same scene type. For example, it may be possible to define a scene type Travel Info which both Hotel Info and Travel Info are instances of. However the message contents (as well as the roles) must be the same, if scenes are of the same type. As the information required about flights is quite different than that required for hotels we decide not to generalize to a Travel Info scene type as it would preclude us from having the flight/hotel specific structure in the messages.

Step 3: Develop the performative structure (network of scenes) and initial flow. Having defined what scenes exist, based on the scenarios, we now develop the performative structure which shows how the scenes are linked up and how agents "flow" between the scenes. Additionally, we define which roles play parts in which scenes (initial information is based on the scenarios), and specify how many instances of the roles can take part in a scene instance. For example, one of the scenes is Flight Info. This scene involves the roles of Customer and Flight Provider, with potentially many Customers, but exactly one Flight Provider. We define the minimum number of Customer roles to be 0, and the maximum 1, while both minimum and maximum for the Flight Provider role are defined as 1.

In order to obtain the flows between scenes we can start by mapping the flow implied within our scenarios from the previous phase. We then visit each scene in turn to determine where else each agent might go, from that scene, other than what was captured in the scenario. In the particular scenario we had developed a Customer and Flight Provider start off (after entry) in the Flight Info scene. In the following step, the

Customer is in the Hotel Info scene, implying that the Customer can move from Flight Info to Hotel Info. In generalizing we recognize that the customer can move back and forth between these two scenes, and could in fact come to either of them after the entry scene. Scenario variation descriptions from the system specification phase may provide information regarding additional flows. When specifying how a role can move between scenes, we must also consider whether they will go to one, some, or all instances of that scene. For example we have a Flight Info scene for each provider, so a Customer may well choose to go (simultaneously) to multiple scene instances. Therefore we choose *some* as the specification.

By default if a role can transition to multiple scenes, we use an OR connector. If there is only a single choice, we could make it either AND or OR. However we choose OR, in order to highlight any actual cases of AND which are more unusual.

As we define the flows it is sometimes necessary to introduce new roles. For example, in defining the flow into the Booking scene, we need the Customer to be able to be accompanied by either the Flight Provider (coming from the Flight Info scene), or the Hotel Provider (coming from the Hotel Info scene). As the flight and hotel providers will play the same role within the Booking scene, we need to introduce a new role, which each of them can transform into. We introduce the Booking Provider role, which is added to the role structure. We also introduce the role Payer at this stage for the Banking scene, which the Customer transforms into, as it is somewhat more generic.

As we develop the performative structure it sometimes becomes quite complex, in which case it can be advantageous to abstract a part of it and have nested performative structures. Figure 5 shows the performative structure developed for our example. It could be an option to abstract the structure between flightinfo, hotelinfo and booking into a sub-performative structure (which is actually the scenario structure identified at the top level in the analysis overview).

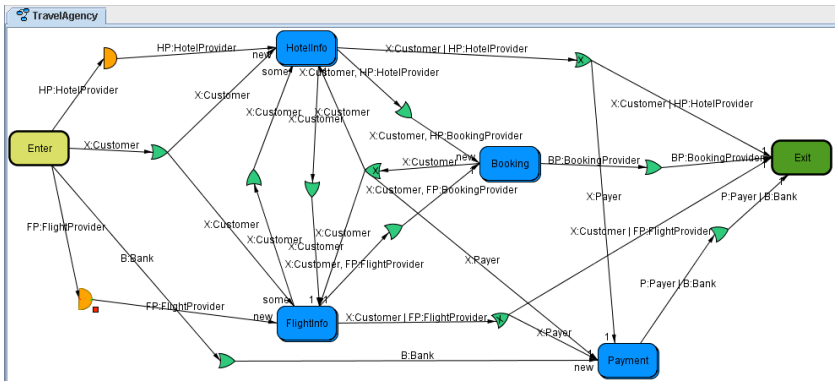

Fig. 5. Performative Structure (from Islander)

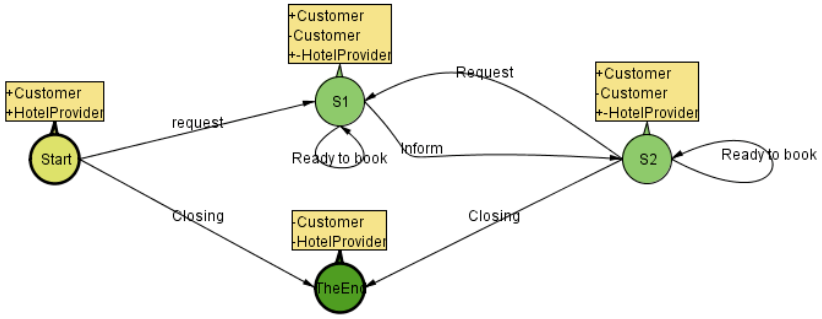

Fig. 6. Hotel Info Scene (from Islander)

Step 4: For each scene, define the interaction structure. The next step involves development of the details of the interaction within a particular scene. The representation used here is a directed graph, which can be seen as an annotated finite state machine. Transitions between states are messages, with the contents defined according to the ontology model. Consequently there is substantial interaction with step 5, development of the ontology. Each state is also annotated with information as to which role types can enter ("+") or leave ("−") at that state. There is also a "stay-and-go" ("+−") annotation which allows an agent to simultaneously be in multiple scenes. For example, when a Flight Provider leaves a scene, with a Customer, in order to make a booking (in the Booking scene), it also stays within the Flight Info scene to attend to other customers.

Figure 6 shows the annotated states and message transitions for the Hotel Info scene in our example. There are some issues to determine in setting up when a Customer may leave the scene. If we wished to require that a Customer waited to receive the response to a request, before leaving, we would not allow them to leave at S1. However this would have the effect that no Customer would be able to leave while any Customer was awaiting a response. If finding information took some time, this could cause unnecessary delays. Consequently we allow a Customer to leave without waiting to receive a response, if desired. Customers tell Hotel Providers when they are ready to go and book so both can ask the infrastructure to leave and enact a booking scene. The scene ends when the Hotel Provider sends a closing message, as customers can come and go as desired.

Part of this step also involves defining the structure of the relevant messages, which is done in Islander by opening a message specification window, where we specify the illocutionary particle, sender, receiver, and message content. For example, the left-most *request* arc may be specified as being from a Customer to a Hotel Provider, and containing a location, desired check in and check out dates, and the class of hotel sought.

Step 5: Define the ontology. There is some initial ontological information specified during the System Specification phase, as one identifies the information needed within scenarios. This can be brought into the Social Design and provides the basis for more

thorough refinement and development. In our example the types of data that have been identified are Hotel Info, Flight Info and Booking. We determine that Booking really needs to be specialized into Flight Booking and Hotel Booking, so these are added to the ontology.

As messages are developed in a scene, this typically results in further additions to the ontology. For example when defining the messages in the Hotel Info scene we recognize the need for a Hotel Info data type and add this into the ontology.

Step 6: Define the information model. Having defined the details of scenes, we also need to specify what information needs to be maintained within the system, for use either by the institution, or by the agents filling the roles. All information that will be referenced within constraints or norms, needs to be part of the information model. Actions then need to be defined for the roles which modify and access this information model. For each property in the information model it must be considered whether the information should be defined per role, or per institution.

One particular type of information that needs to be maintained in our example is the bookings that a customer has, and the payments which are due for those bookings. This is clearly information which is required for each Customer role. Consequently we create Payment Due, as a list of Payment Details, within the Customer role. We then add actions that update this. For example an accept in the Booking scene causes the action to add the payment to the list of payments due for that Customer. In the Payment scene when a payment is successful (i.e. confirmed by the Bank) this results in an action to remove that payment from the list.

Constraints are also added at this stage. An example constraint in our model is that a Customer can only request to make a Payment that is in its Payments Due list.

Step 7: Identify and specify norms. Norms are conditions that should be ensured by the infrastructure of the institution. They are specified in Islander, in terms of an antecedent utterance which triggers the commitments associated with the norm, a consequent which captures the commitment or obligation, and an utterance (called the defeasible antecedent) which specifies when the commitment is regarded as being fulfilled. The norms are usually defined towards the end of the Social Design process when the infrastructure is fully defined. An example norm from our travel institution is that if a Booking is made, the agent cannot leave the institution until the corresponding payment is made.

Step 8: Check all social goals are achieved. Finally we check through all of the social goals to ensure that some aspect of the institution does ensure that these are met. In our example one of the social goals that was identified was ensuring that service providers were reliable. This is not something that can be specified by a norm or constraint based on the current specification. One solution may be to introduce a scene where customers can make complaints which will be maintained by the institution, and any provider having too many complaints could then be banned from entering the institution, thus providing some level of realization of this social goal.

Iteration through the Islander models: The steps described should ensure a thorough design of the electronic institution within Islander. They cover each of the Islander models: the Dialogical Framework with the roles and ontology, the Performative Structure that captures the scenes and transitions between them, the Interaction Model which specifies the allowable communication patterns within each scene, and the Norms and Commitments.

3.3 Architectural Design

The main tasks of the Architectural Design in Prometheus are to determine the agent types, to specify various details regarding these, and to develop protocol specifications, including messages. This results in a system overview diagram which gives an overview of the agents, the interactions between them, and the interface to the environment (in terms of percepts and actions); as well as *interaction protocols* specified in AUML[7], which are developed by refining scenarios to give interaction diagrams, which in turn are generalized to provide protocol specifications. As part of developing the protocols, the messages between agents are also specified.

The decision as to which roles to combine into agents is based on standard software engineering concepts of cohesion, coupling and modularity. The agent goals, along with the percepts they receive and the actions they execute, are then propagated from the system specification. In addition the developer is prompted to consider a range of questions regarding initialization, cardinality, and other aspects of the agent.

This step, of deciding how to group roles into agents, remains essentially unchanged, except that the Social Design phase may well influence this. We do add a field in the agent descriptor to allow specification of norms that apply to an agent. This information will then be passed to the detailed design where agent behaviour should be developed to respect these norms.

The introduction of the Social Design phase means that a large part of the message and interaction specification has already been done. We note though that there is some difference in that the interactions of the Social Design phase are between roles, whereas standard Prometheus design specifies interactions between agent types. If the choice is made to implement an agent type for each role in the Social Design there is no difference. However, if some roles are combined into a single agent type, then some adjustments may be needed.

For example in our travel institution we may decide to combine the Flight Provider and the Hotel Provider into a single Travel Agent agent type. This combination is straightforward in that none of the specified interactions are between Flight Provider and Hotel Provider. Consequently we can just replace Flight Provider and Hotel Provider by Travel Agent as we convert the interaction specified in the scene, to an AUML protocol.[8] In cases where there are potential interactions between roles that have been incorporated into the same agent type, we must ensure that we do still specify the interactions that may happen as a result of two agent instances of this type, playing different roles.

[7] http://www.auml.org

[8] This can likely be automated, but we have not yet done this, nor investigated it fully.

The ontology of the Social Design phase can be incorporated directly into the Architectural Design, and once the mapping between roles and agents is clear, it is obvious which agents deal with which data.

It may be the case that not all agents in the system being developed participate in the electronic institution. If this is the case standard Prometheus process needs to be followed to develop the interaction protocols involving these agents. In some cases there may be a situation where an agent is interacting with another agent outside the EI, within the same protocol that it is interacting within the EI. From the point of view of the EI, these additional interactions are part of the agent internals. But from the point of view of the entire system they are part of a larger interaction protocol. For example if our system included an agent, outside of the EI, whose job it was to continually search the Internet for good flight deals within Europe, then our flight provider may well interact with this agent to get up to date information, between the request and the inform within the Islander flight info scene.

Consequently, although a substantial amount of information can be incorporated from the Social Design into the Architectural Design, it is still necessary to consider all the Prometheus steps, in the case that there are some parts of the system that do not participate in the electronic Institution.

4 Related Work

Engineering multi agent systems (MAS) is an intricate task that sits on top of disciplines like distributed and normative systems, and that frequently uses metaphors from the social sciences (e.g. sociology, economy, or psychology). It would therefore be lengthy to try and make a complete summary of the state of the art that covers all the sources of influence. We will therefore concentrate on work that is more directly related to MAS organizations and software development methodologies.

The organization of a MAS consists of the roles, relationships, and power and authority relationships among the roles that structure the behaviour of the agents. For any agent, the access permissions, actions allowed, and interactions permitted depend to a large extent on what roles the agent might incarnate within an organization. For instance, the organization associated to an electronic institution is called its *social model* and is specified as a set or roles, a role hierarchy, and user-defined relationships. The actions permitted are determined by some system-defined relationships (static and dynamic separation of duties), by the role flow policy in the performative structure, and by the protocols within scenes.

All MAS have an organization of some sort underpinning them, and in the literature of agents there is a large corpus of work devoted to studying the algorithmics and the problem solving capabilities that different organizations may show. Some of the most-studied organizational structures are *hierarchies*, *coalitions* and *teams*.

Hierarchies [14] are the most primitive, where the tree of the hierarchy determines the interactions that might happen (between parents and children only) and thus how the information flows (up and down), and the authority relationship (top-down). Electronic institutions could be embedded with hierarchical organizations if care is taken in the performative structure to only allow the type of interactions that the hierarchy

establishes. The authority relationship is mapped easily by the hierarchy defined in the social model.

Coalitions [15] are organizations that are much more dynamic in the sense that the structure is not fixed at specification time but it is an 'agreement' that agents commit to in order to act in a co-ordinated way. Coalitions need therefore to be formed at run time upon a certain common goal. Algorithms to determine the optimal coalition structure for a problem have been studied [16].

Teams [17], like coalitions, are dynamically organized groups of agents that have different individual goals but that co-operate to attain a certain global goal that requires the concourse of all of the members of the team. In both cases, coalitions and teams, the institutional perspective is that of laying down the infrastructure that would permit the dialogues and commitments among the agents (together perhaps with norms that would punish the violation of agreements).

A large number of MAS software development methodologies have been proposed recently (e.g. see [18, 19]). Although they are based on strong agent-oriented foundations and offer original contributions at the design level, they are unsatisfactory for developing EIs: most MAS methodologies, although they necessarily deal with structures of agents and interations between agents, do not explicitly represent community or social concepts. More generally, the formal definition of organization-centered patterns and social structures in general (e.g. [5, 20, 21]), and their computational realization remain open issues (as noted in [22]).

This work provides a detailed methodology for developing an explicit institutional structure, and embeds this into an existing MAS methodology. The integration within the Prometheus methodology means that Prometheus (and PDT) can then be used to design and develop the agents that will participate in the specified Electronic Institution.

Our approach shares some similarities with the OperA methodology [5]. OperA builds upon the idea of an organizational model, consisting of roles and their relationships, similar to those we use, and an interaction model inspired by the electronic institution concept, that determines the activities agents get engaged in. Norms are non-operational concepts in OperA that describe the behaviour of agents in an abstract way. In our approach we opt for a more grounded approach that permits certain verifications of agent behaviour. Finally, OperA defines the initial part of the interaction network (start scene) as the setting of a social model where agents agree on social contracts that later on they will freely respect in their interactions.

Although some agent infrastructures such as DARPA COABS[9] and FIPA compliant platforms such as JADE [23] deal with many issues that are essential for open agent interactions — communication, identification, synchronization, matchmaking — they are arguably too distant from organization-centered patterns or social structures. Also, although some infrastructure work, perhaps most notably the work on TuCSoN [24], has investigated linking lower-level infrastructure with social laws (e.g. [25]), clear methodological guidance for a designer has not been well addressed.

Among the few other proposals we can mention the proposal by Hanachi [26] that allows for specifications of interaction protocols that need to be subsequently compiled into a sort of executable protocol brokers called moderators. Also, in Tropos, the specifi-

[9] http://coabs.globalinfotek.com/

cations are transformed into agent skeletons that must be extended with code. However, at execution time there is no mechanism to ensure that agents follow the specification of the system.

A promising line of work is the one adopted by Omicini and Castelfranchi (e.g. [27]). It postulates some significant similarities with the EI approach: focus on the social aspects of the interactions, a unified metaphor that prevails along the development cycle, and the construction of tools to implement methodological ideas. However, the actual development of the methodology and the associated tools appears to be still rather tentative.

5 Conclusion

We have presented a methodology for designing e-institutions that extends the Prometheus methodology with a social design phase, where Islander is used to design an institution.

It appears to be relatively straightforward to actually integrate the two tools (PDT and Islander) by means of XML specifications of entities that are passed between them. This work is currently in progress. We are also investigating developing skeleton code from the Prometheus Detailed Design, which can be integrated into Ameli at runtime.

References

1. Horling, B., Lesser, V.: A survey of multi-agent organizational paradigms. The Knowledge Engineering Review **19** (2005) 281–316
2. Esteva, M.: Electronic Institutions: from specification to development. IIIA PhD Monography. Vol. 19 (2003)
3. Rodríguez-Aguilar, J.A.: On the Design and Construction of Agent-mediated Electronic Institutions. IIIA Phd Monography. Vol. 14 (2001)
4. Noriega, P.: Agent-Mediated Auctions: The Fishmarket Metaphor. IIIA Phd Monography. Vol. 8 (1997)
5. Dignum, V.: A Model for Organizational Interaction. PhD thesis, Dutch Research School for Information and Knowledge Systems (2004) ISBN 90-393-3568-0.
6. Esteva, M., de la Cruz, D., Sierra, C.: Islander: an electronic institutions editor. In: Proceedings of the First International Joint Conference on Autonomous Agents and Multi-agent Systems (AAMAS 2002), Bologna, Italy (2002) 1045–1052
7. Arcos, J.L., Esteva, M., Noriega, P., Rodríguez, J.A., Sierra, C.: Engineering open environments with electronic institutions. Journal on Engineering Applications of Artificial Intelligence **18** (2005) 191204
8. Padgham, L., Winikoff, M.: Developing Intelligent Agent Systems: A Practical Guide. John Wiley and Sons (2004) ISBN 0-470-86120-7.
9. Padgham, L., Thangarajah, J., Winikoff, M.: Tool Support for Agent Development using the Prometheus Methodology. In: First international workshop on Integration of Software Engineering and Agent Technology (ISEAT 2005), Melbourne, Australia (2005)
10. Esteva, M., Rodríguez-Aguilar, J.A., Sierra, C., Arcos, J.L., Garcia, P.: On the formal specification of electronic institutions. In Sierra, C., Dignum, F., eds.: Agent-mediated Electronic Commerce: The European AgentLink Perspective. Number 1991 in Lecture Notes in Artificial Intelligence. Springer-Verlag (2001) 126–147

11. Rodríguez-Aguilar, J.A., Noriega, P., Sierra, C., Padget, J.: Fm96.5 a Java-based Electronic Auction House. In: Second International Conference on The Practical Application of Intelligent Agents and Multi-Agent Technology(PAAM'97). (1997) 207–224
12. Cuní, G., Esteva, M., Garcia, P., Puertas, E., Sierra, C., Solchaga, T.: MASFIT: Multi-agent Systems for Fish Trading. In: 16th European Conference on Artificial Intelligence (ECAI 2004), Valencia, Spain (2004) 710–714
13. van Lamsweerde, A.: Goal-oriented requirements engineering: A guided tour. In: Proceedings of the 5th IEEE International Symposium on Requirements Engineering (RE'01), Toronto (2001) 249–263
14. Fox, M.S.: Organization structuring: Designing large complex software. Technical Report CMU-CS-79-155, Carnegie-Mellon University (1979)
15. Shehory, O., Kraus, S.: Methods for task allocation via agent coalition formation. Artificial Intelligence **101** (1998) 165–200
16. Chvatal, V.: A greedy heuristic for the set covering problem. Mathematics of Operations Research **4** (1979)
17. Tambe, M.: Towards flexible teamwork. Journal of Artificial Intelligence Research **7** (1997) 83–124
18. Henderson-Sellers, B., Giorgini, P., eds.: Agent-Oriented Methodologies. Idea Group Publishing (2005)
19. Bergenti, F., Gleizes, M.P., Zambonelli, F., eds.: Methodologies and Software Engineering for Agent Systems. The Agent-Oriented Software Engineering Handbook. Kluwer Publishing (2004) ISBN 1-4020-8057-3.
20. Parunak, H., Odell, J.: Representing social structures in uml. In: Agent-Oriented Software Engineering II. LNCS 2222. Springer-Verlag (2002) 1–16
21. Vazquez, J., Dignum, F.: Modelling electronic organizations. In: Multi-Agent Systems and Applications III. Volume 2691 of LNAI. Springer-Verlag (2003) 584–593
22. Zambonelli, F., Jennings, N., Wooldridge, M.: Developing multiagent systems: The gaia methodology. ACM Transactions on Software Engineering and Methodology **12** (2003) 317–370
23. Bellifemine, F., Poggi, A., Rimassa, G.: Developing Multi-Agent Systems with JADE. In Castelfranchi, C., Lesperance, Y., eds.: Intelligent Agents VII. Number 1571 in Lecture Notes in Artificial Intelligence. Springer-Verlag (2001) 89–103
24. Cremonini, M., Omicini, A., Zambonelli, F.: Multi-agent systems on the Internet: Extending the scope of coordination towards security and topology. In Garijo, F.J., Boman, M., eds.: Multi-Agent Systems Engineering. Volume 1647 of LNAI., Springer-Verlag (1999) 77–88 9th European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW'99), Valencia, Spain, 30 June – 2 July 1999. Proceedings.
25. Ciancarini, P., Omicini, A., Zambonelli, F.: Multiagent system engineering: The coordination viewpoint. In Jennings, N.R., Lespérance, Y., eds.: Intelligent Agents VI. Agent Theories, Architectures, and Languages. Volume 1757 of LNAI., Springer-Verlag (2000) 250–259 6th International Workshop (ATAL'99), Orlando, FL, USA, 15–17 July 1999. Proceedings.
26. Hanachi, C., Sibertin-Blanc, C.: Protocol Moderators as Active Middle-Agents in Multi-Agent Systems. Journal of Autonomous Agents and Multiagent Systems **8** (2004)
27. Omicini, A., Ricci, A., Viroli, M., Castelfranchi, C., Tummolini, L.: Coordination artifacts: Environment-based coordination for intelligent agents. In: Third International Joint Conference on Autonomous Agents and Multi-agent Systems (AAMAS'04), New York, USA (2004) 286–293