Flexible Coordinator Design for Modeling Resource Sharing in Multi-Agent Systems[1]

Jiexin Lian and Sol M. Shatz
University of Illinois at Chicago
Chicago, IL, 60607

Xudong He
Florida International University
University Park, Miami, FL, 33199

Abstract – One approach to modeling multi-agent systems (MAS) is to employ a method that defines components which describe the local behavior of individual agents, as well as a special component, called a *coordinator*. The coordinator component coordinates the resource-sharing behavior among the agents. The agent models define a set of local plans, and the combination of local plans and a coordinator defines a system's global plan. Although earlier work has provided the base functionality needed to synthesize inter-agent resource sharing behavior for a global, conflict-free MAS environment, the lack of coordination flexibility limits the modeling capability at both the local plan level and the global plan level. In this paper, we describe a flexible design method that supports a range of coordinator components. The method defines four levels of coordination and an associated four-step coordinator generation process, which allows for the design of coordinators with increasing capabilities for handling complexity associated with resource coordination. Colored Petri Net-based simulation is used to analyze various properties that derive from different coordinators and synthesis of a reduced coordinator component is discussed for cases that involve homogeneous agents.

Keywords: Multi-Agent System, Colored Petri Net, Conflict Resolution, Coordinator Design, Design Modeling, Simulation.

1. Introduction

Multi-agent system (MAS) design has emerged as a powerful approach to perform tasks or solve problems in a decentralized environment [KG97]. Many of the technologies supporting MASs stem from distributed artificial intelligence research [GH97] and such systems are expected to solve problems that may be too large for a single agent, to provide enhanced speed and reliability, and to tolerate uncertain data and knowledge [GH97].

A MAS can be characterized by the following properties [JS98]: (1) Each agent has incomplete capabilities to solve a problem; (2) there is no global system control; (3) data is decentralized; and (4) computation is asynchronous. These properties imply that agent coordination is a critical aspect of MAS design [J96], which motivated the development of a modeling method for conflict control in such agent-based systems [LS07]. That method embraces the principle of "separation of concerns" in the agent-oriented design and focuses on modeling the possible behaviors of the MAS. As such, the work is complementary to other research efforts that model other

[1] This material is based upon work supported by the U. S. Army Research Office under grant number W911NF-05-1-0573 (Lian, Shatz) and also work partially supported by NSF grant HRD-0833093 (He).

aspects of a multi-agent system [XS03, HM04]. Using this method, a designer first creates a local plan for each agent, and then forms a unit called a coordinator to integrate the inter-agent resource sharing behaviors from the local plans. Finally, the local plans and the coordinator are concatenated to build a global plan – an integrated model for the whole MAS system. Local plans include statically defined multiple paths for agent actions. When an agent is executed according to a path in its local plan and a conflict arises, an alternative path can be selected from the local plan in order to continue the execution. Because we avoid the need to dynamically derive an alternative path (although the path selection is dynamic), higher temporal efficiency can be achieved. Also, since the paths of individual agents are determined in the design stage, agent behaviors can be predicted and analyzed to achieve better resource utilization. Alternative path selection is decided by the global plan and is controlled by the coordinator component.

To implement a fully-featured MAS, our agent behavior model needs to be integrated with other aspects of an agent. Our intent is not to force a MAS designer to represent all aspects of a MAS within a Petri net based formalism; we simply use net modeling as a formalism for focusing on agent coordination. Our previous work [LS08] gave some guidelines on integrating agent-based Petri net models with the widely accepted BDI agent model [KG96]. In a BDI agent model, an agent has goal, plan, knowledge base, and environment modules. The way agents accomplish a goal task is specified by agent plans built from capable basic agent actions. The local plan of an agent can be integrated into the BDI agent as the agent plan. Coupled with the agent goal, its knowledge-base, environment and reasoning mechanism, agents can autonomously select proper actions to achieve the global goal. Also, performing an action will update an agent's mental state.

It is worth noting that the integrated global model we use in this work only exists during the modeling stage and is for the purpose of defining a conceptual model that can support design analysis. We do not want to imply any constraint on how that design model is later realized during the implementation stage – in fact we would expect that the conceptual centralized coordinator model may indeed be realized by a distributed implementation. Each local plan serves as a high level specification for an individual agent and gets implemented as a self-contained software artifact. The implementation for each agent may vary, as long as its behavior is captured by the local plan. Communication between self-contained agent implementations is defined by potential arcs, which can be translated into different agent communication protocols. For example, Service Oriented Architecture [NL05] can be adopted as a practical implementation framework, where each local plan can be implemented as a software service, and the potential arc can be translated into a communication protocol using Web Service Definition Language (WSDL). Furthermore, a modeling tool such as CPN Tools [CPN06] can generate executable code from Colored Petri nets. Thus this may serve as a basis for synthesis of agent code. Clearly, there are many research topics in the area of MAS implementation. Since the focus of this paper is on model creation and analysis, further details related to implementation are not discussed.

Our earlier work provided the base functionality needed to synthesize inter-agent resource sharing behavior in a global and conflict-free MAS environment [LS07]. But in that work, the lack of coordination flexibility limits the modeling capability at both the local-plan level and the global-plan level. At the local-plan level, the base coordinator is formed by simply matching the resource types between the relevant resource supplies and demands,

but many important resource coordination constraints cannot be expressed directly by local plans. As a consequence, the coordinator is unable to satisfy these coordination constraints. At the global level, because the global MAS design strategy may differ from the local agent interests, it is highly desirable to provide the flexibility of incorporating some form of global strategies into the MAS design in a standard and consistent format.

In this paper we propose a new coordinator design method that allows for progressively enhancing the flexibility of coordination. The kernel of this design method is a four-step design framework that provides the MAS designers a spectrum of tools to improve coordinator flexibility by imposing increasingly rigorous resource sharing constraints, from both local and global perspectives. These resource sharing constraints can alleviate limitations inherent in the models created using the approach of [LS07], and allow the coordinator, as a distinct entity, to control the inter-agent resource sharing behavior for various purposes. As a result, we can obtain a set of coordinators with different levels of flexibility, and each coordinator can be used to generate a global MAS model featuring different resource sharing constraints. While it may appear that the coordinator component is a centralized hub for resource sharing, this is simply an abstraction property due to the nature of global plan models. For an implementation based on this model, the coordinator can be divided into sub-models encapsulated into distributed software modules. In other words, the centralization at the modeling level provides a panorama for the overall coordination, but an implementation is not obligated to follow a centralized architecture.

We use Colored Petri net and the extended "potential arc" concept first introduced by [LS07] as our modeling notation. Potential arcs serve to extend the traditional CPN model with explicit support for distinguishing the representation of potential conflicts and real conflicts, and thereby avoiding the need to explicitly eliminate all potential conflicts in the design stage. This allows for independence in local-plan designs, i.e., independence among the local behavioral plans for individual agents. The modeling notation is based on Colored Petri nets (CPN) because CPN-based MAS models can provide for simulation of the dynamic execution of a MAS and the token driven mechanism inherent to CPN modeling can be used to describe the event triggering mechanism of conflicts. Also, the existence of other CPN-based agent modeling techniques [MW97, HM04, HB04, and XS03] indicates that colored Petri nets have properties that make them an effective tool in MASs modeling. However, the coordinator design method advocated in this paper does not have to be bound to any specific modeling notation. Also, to our knowledge, previous efforts on using Petri nets in modeling multi agent interactions did not deal with explicit coordinator design in the systematic way that is presented in this paper.

It is worth noting that the potential arc notation does not expand the modeling capability of traditional Colored Petri nets, nor does it guarantee the generation of optimal (sized) models. Instead, potential arcs make "separation of concerns" applicable to multi-agent system designs, possibly at the cost of increasing model complexity. With potential arcs, multi-agent system model generation can be decomposed into model components and easily automated by algorithmic processes, thus supporting model generation for complex scenarios. Model optimization (following model generation) is not a main concerned here, and can be one future research topic.

Although the primary focus of this paper is on system modeling, it should be noted that the MAS model can be analyzed at both the local plan level and the global plan level. With regard to the local plan analysis, one can ignore potential inscriptions and analyze the local plan as a regular CPN. This is a type of "optimistic" analysis,

based on the following assumption: all resources requested by agents are satisfied during the life-cycle of the agent. The local plan analysis results – for example, occurrence graph, boundness and liveness – still capture the agent's local characteristics. Further details on local plan analysis are beyond the scope of this paper. Since a created global plan is also a CPN, global plan analysis can be used to capture system-wide properties. Particularly, since potential arcs are used to model the resource demand locally (so that we can distinguish potential conflicts and real conflicts), one can analyze a global plan to study how these real conflicts are resolved and the relevant resource utilization pattern from the global perspective.

The rest of the paper is organized as follows. Section 2 provides background information. This section begins with a summary of Colored Petri Net Terminology used in the rest of this paper. After that, we also outline the potential arc based MAS modeling method, previously published [LS07]. Section 3 presents a four-level coordinator design method that emphasizes coordination flexibility and Section 4 discusses model property verification. Finally, Section 5 provides a conclusion and mentions future work.

2. Background and Related Work

The focus of this paper is on designing flexible coordinator components that are compatible with a general MAS modeling method based on Petri net notation [LS07]. For background information, Section 2.1 reviews some key aspects of colored Petri net (CPN) terminology, as introduced in [J97], and Section 2.2 reviews the basic features of the general MAS modeling method. Readers with background in CPN-based modeling can ignore Section 2.1. Section 2.3 discusses related research in multi-agent coordination and communication.

2.1 Colored Petri Net Terminology

As an extension of basic Petri net [M89], colored Petri net (CPN) models are mathematically precise models, allowing both the structure and the behavior of Petri net models to be described using mathematical concepts. We assume that the reader has some familiarity with basic Petri nets, and we cite some terminologies from [J97] that are important for understanding the details of the modeling notation used in this paper. All the notions and definitions marked by "•" are adopted from [J97]. Also, some simple examples are included by us to illustrate these notations and definitions.

• The terms: *type*, *value*, *operation*, *expression*, *variable*, *binding* and *evaluation* are used in exactly the same way as these concepts are used in functional programming languages.

• The concept*, multi-set* [J97] is defined as follows:

"A multi-set *m*, over a non-empty set *S*, is a function $m \in [S \rightarrow N]$ (N denotes the set of all non-negative integers) which can also be represented as a formal sum:

$$\sum_{s \in S} m(s)`s$$

where S_{MS} denote the set of all multi-sets over *S*. The non-negative integers $\{m(s) \mid s \in S\}$ are the coefficients of the multi-set. $s \in m$ iff $m(s) \neq 0$."[J97]

For example, for set *S = {a, b}*, *m* is a multi-set, where *m(a) = 1* and *m(b) = 2*. This multi-set *m* is also written as *1`a + 2`b.*

Also Ø is used to denote the empty multi-set. An empty multi-set over *S = {a, b} is 0`a+0`b* = Ø.

• "A *type* is a finite or infinite set, and is used as the meaning in functional programming languages."[J97] For example, *integer* is a type, which is an infinite set, as defined in major functional programming languages.

Example: *P* = {weak, strong} is also a type, which is a finite set with two elements. In CPN, a type is also called a *color set*. Therefore, when *P* is used in CPN, it is also called a color set.

• The *elements of a type T*. "The set of all elements in *T* is denoted by the type name *T* itself"[J97]. For example, all elements in integer are denoted by the type name integer.

• The *type of a variable*, v – denoted by *Type(v)*. The notation *Type(v)* is also extended to $Type(A) = \{Type(v) \mid v \in A\}$, where A is a set.

Example: Define two variables *x, y* for color set *P* given above. Define variable *z* for color set integer. So we get *Type(x) = P*, and *Type (z)* = integer. Set *A = {x, y}*, then *Type(A) = {P}*. Set *A' = {x, y, z}*, so *Type(A') = P ∪ integer*.

• The *type of an expression*, *expr* – denoted by *Type(expr)*.

Example: Define *expr = 1`x + 2 `y* as an expression, *Type(expr) = P*.

• The *set of variables in an expression*, expr – denoted by *Var(expr)*.

Example: *Var(expr) = {x, y}* **.**

• A *binding* of a set of variables, *V* – associating with each variable *v ∈ Var(expr)* an element *b(v) ∈ Type(v)*. Also, by *B(t)***,** we denote the set of all bindings for t.

Example: Define a binding *b* for {x, y} as: *(x, y) = (weak, strong)*.

• The value obtained by *evaluating an expression*, expr, in a binding, b – denoted by *expr<b>*.

Var(expr) is required to be a subset of the variables of *b*. And the evaluation is performed by substituting for each variable *v Var(expr)***.**

Example: *expr<b> = 1`weak + 2`strong.* Here *Var(expr) = {x, y}*, and the set of variables in *b* is also *{x, y}*, so the evaluation can be performed. If the set of variables in *b* is *{x}*, we cannot perform evaluation on *expr<b>*, since *{x, y}* is not a subset of *{x}*.

• *B* denotes the boolean type (containing the elements {false, true}).

• "An expression without variables is said to be a *closed expression*. It can be evaluated in all bindings, and all evaluations give the same value –a closed expression is often denoted by the expression itself." [J97] This means that we can simply write "expr" instead of "expr<b>."

Example: *1`weak + 2`strong* is a closed expression.

Based on the terminology above, a CPN is defined by [J97] as a nine-tuple (Σ, P, T, A, N, C, G, E, I), where:

"(1) *Σ* is a finite set of color sets.

(2) *P* is a finite set of places.

(3) *T* is a finite set of transitions.

(4) *A* is a finite set of arcs such that: $P \cap T = P \cap A = T \cap A = \emptyset$.

(5) *N* is a node function. It is defined from *A* into $P \times T \cup T \times P$. (i.e., $A \rightarrow P \times T \cup T \times P$**).** Here the node function defines how places and transitions are connected.

(6) *C* is a color function. It is defined from *P* into Σ (i.e. $P \rightarrow \Sigma$). C assigns color set for each place in *P*.

(7) *G* is a guard function. It is defined from *T* into expressions (i.e. associate an expression with each transition in *T*) such that: $\forall t \in T : [\mathrm{Type}(G(t)) = B \wedge \mathrm{Type}(\mathrm{Var}(G(t))) \subseteq \Sigma]$.

(8) *E* is an arc expression function. It is defined from *A* into expressions (i.e., associate an expression with each arc in *A*) such that: $\forall a \in A : [\mathrm{Type}(E(a)) = C(p)_{MS} \wedge \mathrm{Type}(\mathrm{Var}(E(a))) \subseteq \Sigma]$, where *p* is the place of *N(a)*.

E specifies the inscription for each arc, and these inscriptions are used to update the agent's mental state after a transition is fired.

(9) *I* is an initialization function. It is defined from P into *closed expressions* (i.e., associate an closed expression with each place in *P*) such that $\forall p \in P$: [Type(I(p)) = $C(p)_{MS}$]."[J97]

Note: a *CPN marking* is a function defined from P into *closed expressions* (i.e., associate an closed expression with each place in *P*) such that $\forall p \in P$: [Type(I(p)) = $C(p)_{MS}$]. Therefore, *I* is a special marking to denote the initial status of the CPN, it is also called an *initial marking*.

Related to the definition of CPN, the following concepts are also important for understanding the contents of this paper:

• "A *token tk* for a CPN (Σ, P, T, A, N, C, G, E, I) is an element of a color set *S*, and exists place *P*, and *C(P) = S"[J97]*.

• *A* transition is *enabled* when each of its source places contains more tokens than the corresponding incoming arc inscription.

• If an enabled transition is *fired*, tokens are removed from its source places and deposited into its destination places. The quantity of tokens removed and deposited are specified by the incoming arc inscription and outgoing arc inscription, respectively (Details about the transition enablement and firing can be found in [J97].)

2.2 MAS Modeling Overview

MAS modeling plays a crucial role in Agent Oriented Software Engineering (AOSE). It has attracted much research attention. AOSE methodologies normally divide the agent development process into three major phases – system specification, architecture design, and detailed design. Prometheus [WP04] and Gaia [ZJW03] are two influential AOSE methodologies. Prometheus is an informal framework for developing agent systems. Heuristics were proposed in developing specification and design, and several UML notations such as use case and interaction diagrams were used to document the results. The Gaia method [ZJW03] proposed a computational organizational abstraction of a multiple agent system. Various model constructions based on the organizational view were proposed in the above process stages; however specific modeling techniques were not identified. The MAS modeling method in this paper fits the architecture design stage in AOSE, but it is different from Prometheus and Gaia in that both of those approaches are generic multi-agent system development methodologies that cover a much broader scope than

the coordinator design problem that is considered in this paper. Neither Prometheus nor Gaia explicitly dealt with coordinator specification. [LS07] includes a more comprehensive survey of MAS modeling.

This section reviews the basic features of the general MAS modeling method presented previously. The review provided here is adopted from the overview used as the basis for a case-study in [LSH07]. The method first creates local plans to model each agent's behavior independently, and then concatenates local plans together to form a global plan, which describes the whole MAS system. With regard to modeling an agent's behavior, we want to emphasize the following characteristics of an agent: an agent is an autonomous entity with several properties and actions. An agent's mental state is a possible assignment of the agent's properties, and we abstract an agent's behavior into a set of actions between different mental states. An agent action implements some tasks and updates the agent's mental state. Within an agent, an action is taken through one of several predefined paths. Associated with each path are an agent action and a set of resources that may be acquired or released. In the runtime, an agent action can be taken along any available path, whose availability is determined by the availability of associated resources (especially external resource). The agent is provided the autonomy to select any available path.

To maintain conciseness, an incoming arc refers to the arc from a Petri net place to a transition, while an out-going arc refers to an arc from a transition to a place. To describe paths with external resources that may trigger a conflict, we introduce the potential arc concept. Compared with the basic CPN model, where each arc carries one inscription – referred as regular arcs and regular inscriptions, we now add the capability for an additional inscription to describe the need for access to an external resource that is modeled "out-side" of a local plan. This new inscription is called a potential inscription since the access to the resource is a potential access in terms of the view of the design model. We call an arc that includes both a regular inscription and a potential inscription a potential arc and we call a CPN with potential arcs a Potential Colored Petri Net (PCPN). A potential inscription carried by an incoming potential arc specifies the resource unit required to enable the arc's destination transition, while a potential inscription carried by an out-going potential arc specifies the resource unit released after the arc's source transition is fired. We can view the PCPN model as a superset of CPN models.

For example, a very simple agent AP describes a person's behavior. The person has two properties, isHome and isOffice – to indicate if the person is currently at home or at his or her office. The person also has one action: go-to-work. The action go-to-work moves the person from the home to the office. Both isHome and isOffice have two possible assignments: "YES" or "NO." At a specific run-time moment, isHome equals "YES" means the person is at home, while isHome equals "NO" means the person is not home; similarly for the property isOffice. The tuple (isHome, isOffice) represents the agent's mental state. The state (isHome, isOffice) = ("YES", "NO") means the person is home, and the action go-to-work updates the agent's mental state to ("NO", "YES"). There exist two paths for the action go-to-work: taking a bus or driving a car, where the bus and the car are two different external resources.

We can model agent AP in a MAS environment as follows. Two places Home and Office are defined for the agent's two properties isHome and isOffice, respectively; and transition Move_1 (connected with a potential inscription modeling the external resource bus) and Move_2 (connected with a potential inscription modeling the external resource car) are defined to model two paths for the agent's action go-to-work. The color set for the place

Home can have only one element, since isHome has only two possible assignments, and one assignment corresponds to the default assignment of place Home containing no token. We define the color set as {person}. If the place Home holds the token "person", then isHome equals "YES". The default assignment for isHome can be defined as "NO". Place Office has the same color set as place Home. We also define arcs (Home, Move) and (Move, Office) in the local plan, and the inscriptions in arcs (Home, Move) and (Move, Office) are both "1`person". Fig.2.1 shows the local plan for agent AP. In Fig. 2.1, solid arcs represent regular CPN arcs, while dotted arcs denote the potential arcs. For a potential arc, the inscription above the arc is a regular inscription and the inscription below the arc is a potential inscription. For example, arc (Move_2, Office) is a potential arc, with 1`person denoting the regular inscription, and 1`car denoting the potential inscription - representing the external resource to be released to the external environment.

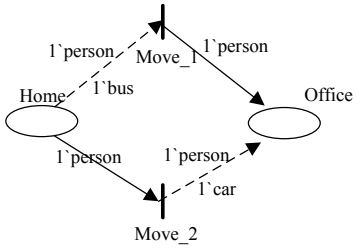


Fig. 2.1 PCPN-based local plan for a simple agent AP

In the run-time, taking the action "go-to-work" can be executed by firing the transition Move_1 or Move_2 in the agent AP's local plan, which results in the token "person" being removed from place Home and deposited into place Office. In order to enable Move_1, we not only need token "person" in place Home, but also some other external agent must provide one resource unit "bus." Enabling Move_2 does not require any external resource (Assuming that we do not consider where the car comes from), but firing the transition will release a resource unit (car) accessible outside the scope of this local plan. The run-time resource availability determines which path will be taken.

After creating the individual PCPN-based local plans for each agent of the MAS, we form a unit called a coordinator to integrate the inter-agent resource sharing behaviors from the local plans. The coordinator is developed by interpreting potential arcs obtained from local plans.

Fig. 2.2 shows the example of how potential arcs are interpreted. In Fig. 2.2, the potential arc (Home, Move_1) from Fig. 2.1 is replaced by a regular arc, and place R1, outside the local plan model, is added to control the enablement of Move_1. R1 needs to produce the token "bus" so that transition Move_1 can be enabled. The interpretation of potential arc (Move_2, Office) is similar. The details of the concatenation algorithm can be found in [LS07].

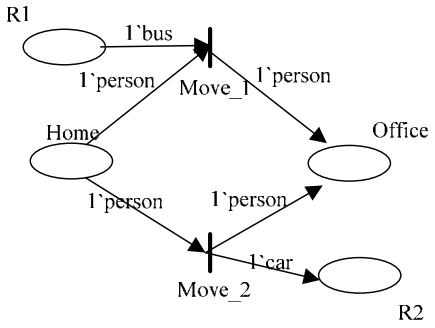


Fig. 2.2 The interpretation of Fig. 1

Finally, the local plans and the coordinator will be concatenated to build a global plan - a CPN model for the whole MAS system. Details of the global plan generation algorithm can be found in [LS07]. Details of the global plan generation algorithm can be found in [LS07]. The created global plan can be analyzed to reason about the behavior of the modeled system. Section 4 presents some experiments on this type of system behavior analysis.

2.3 Multi-Agent Communication and Coordination

As discussed in Section 1, coordination – as the process by which agents reason about and manage the interdependences among their behaviors and try to ensure that all members of the system act consistently [J96] – is critical in MAS design. The easiest way to ensure coherent behavior and resolving conflicts in MAS seems to be granting a specific agent with comprehensive knowledge of the whole system, so that this specific agent can be responsible for coordinating other agents. Such systems typically employ a blackboard architecture to achieve coordination, for instance, the Ambit model for mobile computations [CG98] and the centralized markets – MAGMA [BM96]. As has been noted [GH07], this simplest coordination technique yields classic client/server architecture, and is impractical because it is very difficult to create such a central controller with omniscient knowledge.

Modern coordination mechanisms mainly rely on non-central control schema, including distributed blackboard [CL00], contracting for coordination [S80 and FI02], social laws [ST95 and FT00], and role-based approaches [CF04]. As an extension of the traditional blackboard, Tuple-Spaces [CL00] contain a set of tuples, which hold information relevant for the application. The tuples can be duplicated all over the network. Supported by a programmable interface, information in Tuple-Spaces is retrieved by content of the information. Tuple-Spaces provide the mechanism to guarantee the delivery of information to multiple agents that are unlikely to be concurrently available spatially and temporally, which is a typical case in a mobile agent environment.

Contracting for coordination is a renowned coordination technique for task and resource allocation among agents [S80, and FI02]. [S80] first presented this approach, and [FI02] provides an implementation. In this approach, each agent can be either a manager or a contractor, where the manager announces tasks and the contractor responds to the tasks by bidding. Submitted bids then will be evaluated by the manager for the purpose of contract granting – allocate a task to a specific contractor. This approach provides natural load-balancing but is limited by the fact that does not detect and resolve conflicts [GH97].

Social laws based approaches ask each agent obeys pre-defined social laws to reduce conflicts (similar to vehicles obey traffic laws). Useful social laws set constraints on the agents' activities which allow them to work individually in a mutually compatible manner. [ST95] proposed a general model of social law in a computational system, and investigate some of its properties. [FT00] proposed two basic criteria for selecting among alternative (useful) social laws. However, Social laws based approaches don't address the multi-agent coordination problem from an architectural level.

[CF04] surveyed the existing approaches about multi-agent role-based collaboration and coordination, where a role is defined as a set of common interaction related behaviors that can be applied to an agent in order to change its coordination capabilities. Since all the coordination related behaviors are congregated into a set of roles, the role based approach promotes an organizational view of the system, which well suits agent-oriented approaches [ZJ01]. Also, a role can be applied to various agents with different reasoning mechanisms, so that the role based coordination approach separates the concern of agent reasoning and agent coordination. For this reason, the coordination approach presented in this paper can be classified as a variation of role-based coordination approach, where we specifically focus on conflict control. More importantly, different from all the related work above, we tackle this topic from an architectural level, with a clear intention to perform model analysis.

Other research work has also tackled agent coordination from the architectural perspective. In [S00], Singh proposed a method of using Dooley graphs to generate and synthesize the coordination requirements from multiple autonomous agents. However, the coordination considered in that work focused on the direct communication events among a group of agents, while the coordination in our work addresses the indirect resource sharing among a group of agents. Singh's work dealt with the problem of eliciting and synthesizing coordination requirements, and our work focuses on designing a coordinator component given coordination requirements. Therefore our work and Singh's work are complementary – they address different aspects of coordination and focus on different stages of coordination issues in multi-agent system development.

3. Coordinator Design Based on Progressive Enhancement

Coordinator design is part of MAS modeling. Our MAS modeling method presented in [LS07] can be divided into the following five steps: (1) Acquire modeling requirements and the vocabulary from requirement capture phase; (2) Design the local plan for each agent; (3) Harvest resource requirements from local plans; (4) Design a coordinator; and (5) Generate the global plan. In Step (1), the modeling requirements and the vocabulary are preliminary for model design, and they need to be organized into a specific format. Local plan design (Step 2) and coordinator design (Step 4) generally require human modeling expertise. Step (3) and Step (5) are completely automated (See [LS07] for algorithm details).

Consistent with the MAS modeling method, the following information must first be accumulated to initiate the design of a coordinator: (1) the number of agents in our MAS; and (2) The resource requirements from every agent. Local plans are not used in the coordinator design.

The coordinator defined by our previously published work is fairly naïve since it simply matches and links the resource request and the resource demand from different agents to address the resource supply/demand

relationship between agents. As a natural consequence, the resource is coordinated only by resource type matching. If the same type of resource requested by one agent is provided by another agent, the requesting agent will obtain the resource without further restrictions. Besides resource type matching, it is more desirable for the coordinator to take into account more factors that impact the resource sharing between agents. For example, an agent may prefer one resource type over another when multiple qualified resources are available. These factors can be represented as a category of resource sharing constraints to be included in the coordinator.

We consider a coordinator more flexible if it can automatically coordinate the agent behavior according to more complex resource sharing constraints. In this section, we present a more comprehensive coordinator design method that allows for progressive enhancement of the coordination task, where each step of enhancement encapsulates a specific set of resource sharing constraints in the coordinator. As a result, we can get a different set of coordinators, and each of them can be used to generate a global MAS model for different resource constraints.

Fig. 3.1 illustrates the design procedure, where each step takes into account more information and enhances the flexibility of the coordinator.

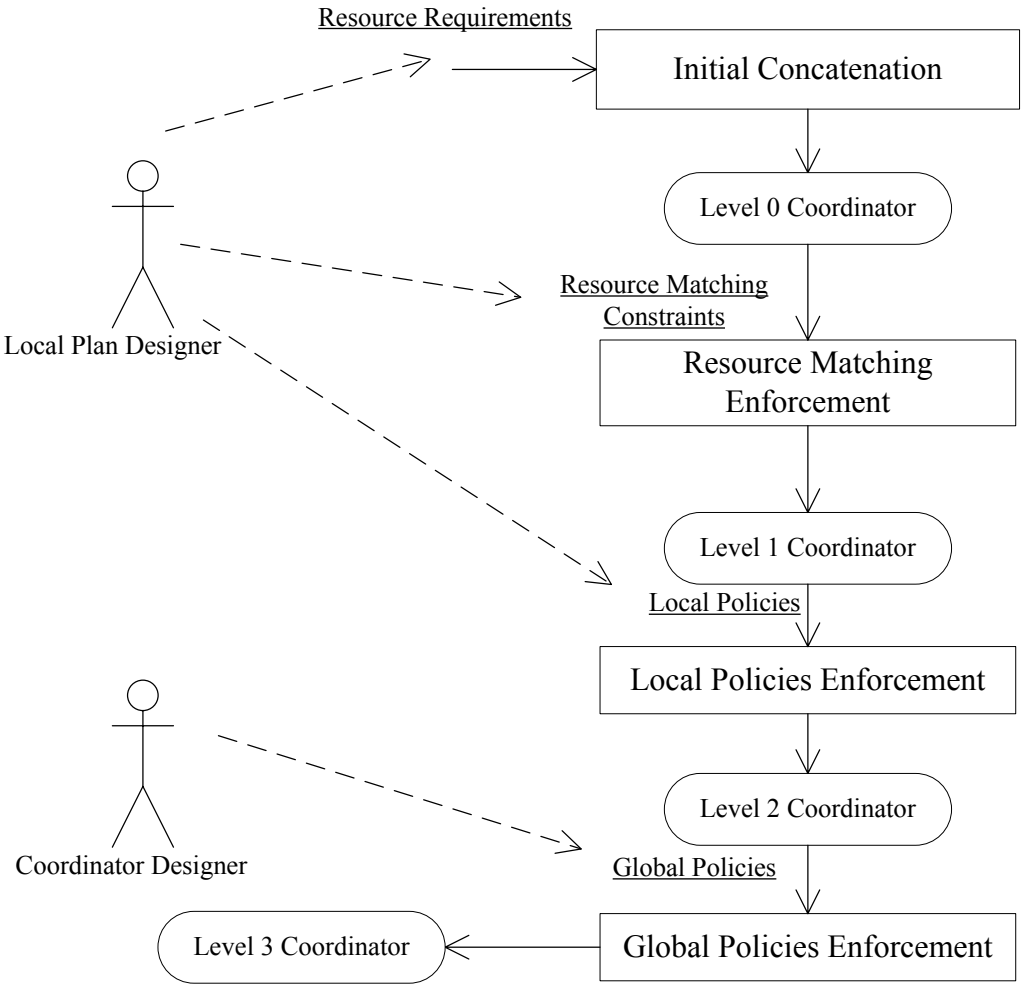


Fig. 3.1 The coordinator design process

Illustrated by Fig 3.1, the coordinator design process can be divided into the following four steps (each represented by a rectangle): (1) Initial concatenation, (2) Resource matching enforcement, (3) Local policies enforcement and (4) Global policies enforcement. After each step, a coordinator with certain capability can be generated, and used to generate the global plan for the MAS. A set of *coordination inputs* are processed at each design step to generate a coordinator. The coordination inputs are information provided to articulate the resource sharing constraints, which take different formats at each design step. Two types of designers are involved in the design procedure. The local plan designer provides three genres of coordination inputs from the perspective of local agents, including resource requirements, resource matching constraints and local policies. The coordinator designer oversees the design from the global level and provides one genre of coordination input: global policies. As a result of coordination input processing, the design procedure generates four coordinators with different levels of capability, varying from level 0 to level 3. The coordinator generated in a certain step is an enhancement of the previously generated coordinator. At each step, a series of standard operations are used to process coordination inputs so that new resource sharing constraints are adopted into the coordinator. For example, after processing the local policies, a Level-1 coordinator is enhanced to level 2, while a Level-1 coordinator is actually an enhancement of a Level-0 coordinator after processing the resource matching constraints. A Level-2 coordinator inherits all Level-1 coordinator resource sharing constraints, which compose a superset for Level-0 coordinator resource sharing constraints. Also, the existing resource sharing constraints of the previously generated coordinator are preserved. Preserving existing resource sharing constraints makes the four design steps relatively independent from each other, and such independence is desirable for design reuse and modification.

This coordinator design procedure provides an *expandable* and *flexible* architecture for the coordinator design. Since the design procedure is based on progressive enhancement, and preserves all the pre-existing resource sharing constraints, designers (the local plan designer and the coordinator designer) can implement a coordinator at any specific coordination level (0-2), and later on still expand that coordinator as more specific design requirements become available. Also, designers have the flexibility to modify inputs caused by the design requirement updates, and we only need to re-process the revised inputs (with the coordinator generated from the immediate previous step) to form a new coordinator. For example, if local policies are modified after the Level-3 coordinator has been generated, we need to re-process local policies with the Level-1 coordinator intact. As a consequence, the Level-2 and Level-3 coordinators will be updated. More importantly, the input processing incurs relatively low implementation cost, since it is carried out by a series of easy automated standard operations.

From another point of view, the four-step design framework provides MAS designers a spectrum of tools to improve the coordinator flexibility by imposing increasingly rigorous resource sharing constraints. In Step 1, the resource requirements harvested from individual agents only contain resource request and demand information, not any other constraints. The Level-0 coordinator simply coordinates inter-agent resource sharing by resource type matching. Stricter resource matching constraints are provided in Step 2 to impose initial resource coordination restrictions. In Step 2, resource matching constraints are used to further limit available resources beyond the limitation defined by the resource type matching. Local policies are incorporated in Step 3 for further enhancement.

Integrating local policies provides the coordinator the ability to handle resource preferences of an individual agent. So, when multiple qualified resources are available, the enhanced coordinator can assign the preferred resource to the agent. Finally in Step 4, the coordinator can incorporate global policies, which are used to enforce some global constraints transparent (unimportant) to local agents.

The rest of this section elaborates our coordinator design procedure. For the purpose of illustration and comparison, we will use a common example to discuss the different levels of coordinators. This running example is a gas Production/Consumption MAS model. This model describes the behavior of a set of gas producers and consumers (each gas producer/consumer is modeled as an agent), with the emphasis on resource sharing. In this MAS, a variety of gasoline products are provided by the producers and consumed by consumers with different demand profiles. The example will be expanded progressively to address increasingly complex design requirements.

3.1 The Level-0 Coordinator

The Level-0 coordinator is identical to the original default coordinator in [LS07], which simply matches and links the resource request and the resource demand from different agents together to address the resource supply/demand relationship between agents. As a natural consequence, the resource is coordinated only by resource type matching. If the same type of resource requested by one agent is provided by another agent, the requesting agent will obtain the resource without further restrictions.

Our MAS starts with two fuel providers: International Fuel Inc. (denoted as IntFuel), which produces regular gas (87 octane) and plus gas (89 octane), and Quality Fuel Inc. (denoted as QualityFuel), which exclusively produces premium gas (93 octane). Two consumers are also included in the MAS. Customer1 only needs to consume gas, but does not worry about the type of gas. In contrast, Customer2 exclusively consumes regular gas. Fig. 3.2 shows the local plans for all the four agents: IntFuel, QualityFuel, Customer1 and Customer2. In Fig. 3.2, *1`()* is a regular inscription, denoting the abstract local resource to drive the production (consumption) action. Potential arcs are denoted by dotted arcs, while the inscriptions below the potential arcs are potential inscriptions (Potential arcs/inscriptions are discussed in Section 2.2).

As shown in Fig. 3.2, two potential arcs are included in the local plan of IntFuel, one for each type of external resources (regular gas and plus gas). Meanwhile, the local plan of QualityFuel only contains one potential arc since there is only one type of external resource, premium gas, involved in the QualityFuel agent. In the local plan of Customer1, one unit of gas of any type is consumed via each pumping action. The local plan of Customer2 is similar to Customer1, with one exception – Customer2 exclusively consumes regular gas. Also, transition *Restart* and local token *1`()* are included in all four local plans to drive the continued production and consumption of gas.

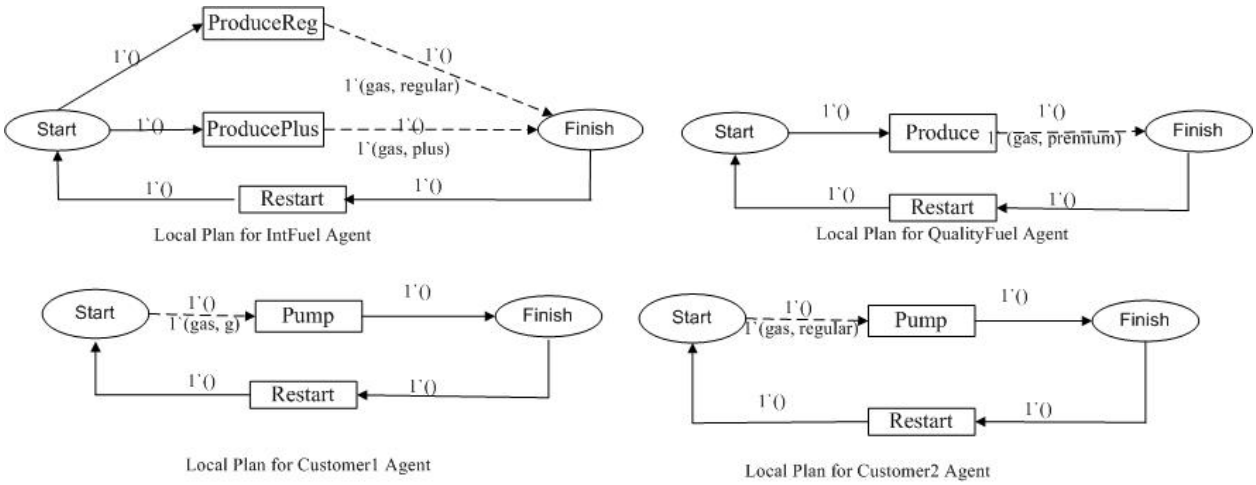


Fig. 3.2 Local plans for two gas providers and two customers

The Level-0 coordinator for the gas production/consumption model is shown in Fig. 3.5. In Fig. 3.5, three types of resources (*regular gas*, *plus gas* and *premium gas*) are all available for the customers, and there are no further resource sharing constraints beyond the resource type matching. To initiate the design of a Level-0 coordinator module, the following information must first be accumulated: (1) the number of agents in our MAS, and the role of each agent; and (2) The resource requirements from every agent. If two agents belong to the same role, they have identical local plans, so the resource requirements for these two agents are the same. Local plans are not used in the coordinator design. The major task in designing a coordinator model is to create transitions to represent agent actions and to connect these transitions with relevant resources, which are represented as CPN places. Through this procedure, a CPN model will be developed.

As indicated previously, the general coordinator design procedure can be divided into the following three steps: (1) global resource requirements coordination, (2) intermediate coordinator generation, and (3) complete coordinator (Level-0 coordinator) generation. Step (1) accumulates the resource requirements provided by individual agents for the convenience of coordinator generation. Guided by the principle of "Separation of Concern," the coordinator generation process is split into two steps. An intermediate coordinator is first created by Step (2) to connect agent actions to resources. Such a coordinator is called an intermediate coordinator in the sense that it may contain unresolved variables. These variables will be resolved in Step (3) to develop the complete coordinator. If there is no unresolved variable in Step (2), the intermediate coordinator is identical to the complete coordinator. Both Step (1) and Step (2) can be automated. In Step (3), an automated algorithm can be invoked to resolve the variables and generate the complete coordinator. We now elaborate on the details of the three steps.

Step 1: *Global resource requirements coordination*. This step simply generates the External Resource Tables (ERTs) to provide a global view of resources that is independent of local plan details. ERT is a data structure defined in [LS07] to harvest the resource requirements and hence support the independence of local plan designs. ERT isolates resource requirements from local places so that the coordinator can be unaware of modifications to a local plan, as long as the ERT remains intact. Each entry in an ERT is a triple *(t, type, PI)*, where *t* is a path transition connected to a potential arc, *type* is *request* or *release*, and *PI* is the potential inscription carried

by a potential arc. Intuitively, an entry *(t, type, PI)* can be interpreted as meaning that the enabling of transition *t* requires external resources specified by inscription *PI* (if type is *request*), or that the firing of transition *t* should release external resources specified by inscription *PI* (if type is *release*).

Fig. 3.3 identifies the ERT data for our *gas production/consumption* example. (Note that the name of an owner-agent (for example, *AP1*) is added as a prefix to the transition name in an ERT entry.) As we can see from Fig. 3.3, Customer1 is demanding gas, not restricted to any specific types (denoted by *(gas, g)*, where *g* is the variable representing any gas type). In contrast, Customer2 exclusively demands regular gas.

ERT Table	Entries
Customer1.ERT	(Customre1.Pump, request, 1`(gas, g))
Customer2.ERT	(Customre2.Pump, request, 1`(gas, regular))
IntFuel.ERT	(IntFuel.RegPro, release, 1`(gas, regular))
IntFuel.ERT	(IntFuel.PlusPro, release, 1`(gas, plus))
QualityFuel.ERT	(QualityFuel.Produce, release, 1`(gas, premium))

Fig. 3.3 ERT for gas production/consumption MAS

Step 2: Intermediate coordinator generation. The generation of the intermediate coordinator can be automated by three key activities: 1) generation of a special CPN place for each resource, called a *resource place*; 2) generation of transitions identified from the ERT data to model agent actions; and 3) generation of arcs to model resource consumption and production relationship. For example, the ERT data identified in Fig. 3.3 can be used to generate the intermediate coordinator in Fig. 3.4. For the entry *(Customre2.Pump, request, 1`(gas, regular))* from *Customer2.ERT* in Fig. 3.4, *p1* is the resource place generated for the resource *(gas, regular), Customre2.Pump* is included in Fig. 3.4 to model the pumping action of customer, and one arc is generated to connect *p1* to *Customer2.Pump*. The arc and its inscription model the following action: *Customer2 consumes a unit regular gas to pump*. *P1* is also connected to *IntFuel.RegPro* by an incoming arc after the entry *(IntFuel.RegPro, release, 1`(gas, regular))* has been processed. Note that there are no initial tokens for this intermediate coordinator, but initial colored tokens can be added later on to emulate the initial condition of the system that is modeled. The generation of intermediate coordinator can be automated, and the automation details can be found in [LS07].

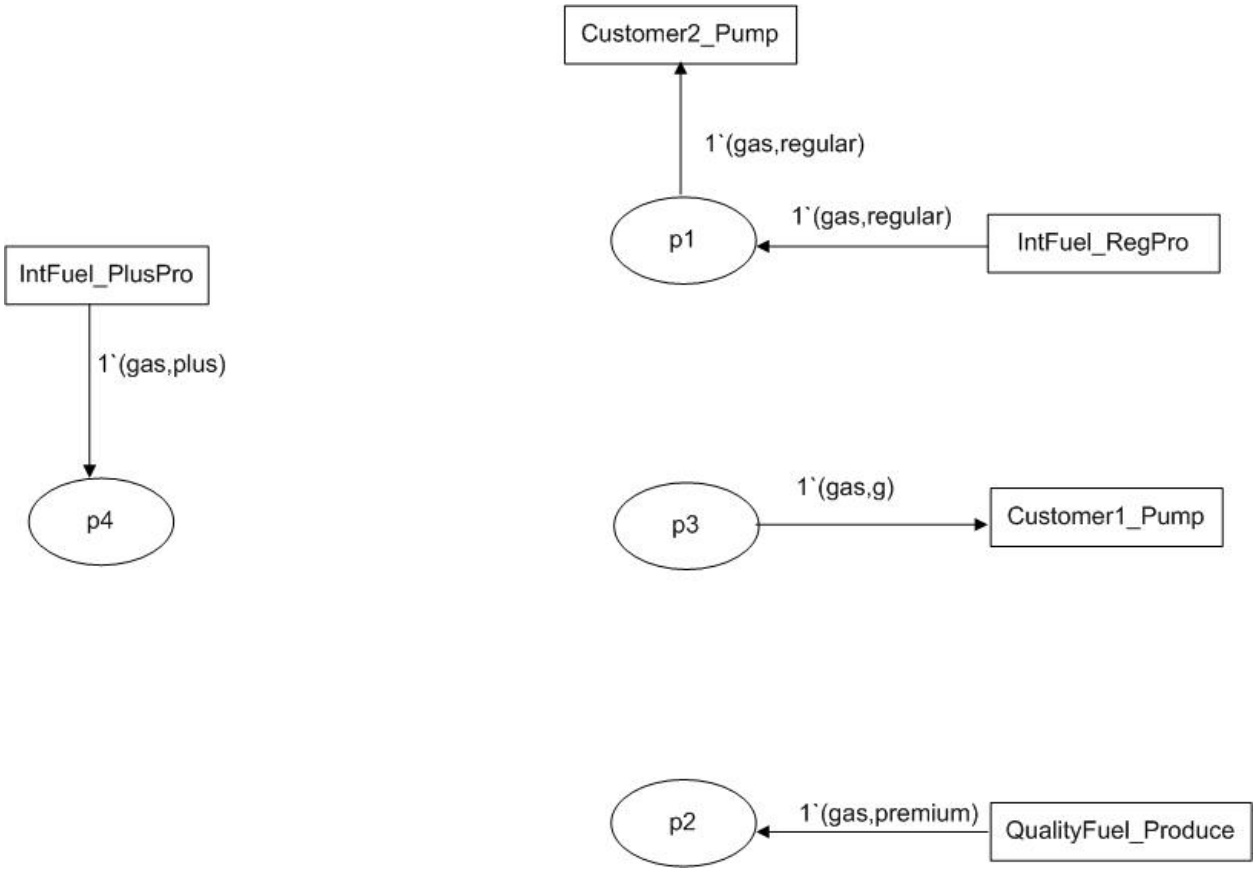


Fig. 3.4 Intermediate coordinator for the gas production/consumption system (variables unresolved)

Step 3: Complete (Level-0) coordinator generation. Since the intermediate coordinator may contain variables, we need to define how variables are resolved in the coordinator. For this reason, we introduce a concept called binding policy in [LS07] to decide which value(s) are allowed to be assigned to a specific variable at the runtime.

The definition of a binding policy is based on a new concept called *token expression*.

Definition - For a CPN (Σ, P, T, A, N, C, G, E, I), a *token expression te* is an expression, where $Type(te) \in \Sigma$. If *te* contains no variable, *te* is a *closed token expression.* According to the definition of multi-set*, closed token expression* is an alias of a multi-set.

Now we can state the definition of a *binding policy*:

Definition - A *binding policy* is a pair *(pre, post)*, where *pre* is a closed token expression over a vocabulary and *post* is a token expression over a vocabulary and contains at least one variable. Further, *pre* and *post* should be token expressions over the same vocabulary.

As a simple example, consider the following binding policy for the gas production/consumption example: *((gas, regular) , (gas, g))*, where *pre = (gas, regular)*, and *post = (gas, g)*. Here *pre* is a closed token expression

(multi-set), and *post* is a token expression, but not a multi-set. Semantically, the binding policy *((gas, regular) , (gas, g))* permits only token color *regular* to be assigned to variable *g*.

After deriving binding policies, the intermediate coordinator can be modified according to the binding policies to form a complete coordinator. In an intermediate coordinator where a binding policy *(pre, post)* applies, there exists a resource place satisfying the token expression *pre* – we call it *pre-place* – and a resource place satisfying a closed token expression *post* – we call it *post-place*. Therefore, binding is implemented by connecting the *pre-place* and the *post-place* by transitions and surrounding arcs, so that a token can be conveyed to a *post-place*, which represents a token expression with variables. We also need to add a transition to allow tokens to be returned to the *pre-place*, in case that these tokens are not consumed from the *post-place*. Based on the analysis above, for each binding policy (pre, post), two transitions – we call them *forward binding transition* (*ft*) and *backward binding transition* (*bt*) – are added into the intermediate coordinator. The following four arcs *a1*, *a2*, *a3*, and *a4* are also be added into the coordinator for each binding policy: *a1 = (pre-place, ft)*, *a2 = (ft, post-place)*, *a3 =(pre-place, bt)*, *a4 = (bt, post-place)*. Inscriptions for all four arcs are the same: *1`pre*.

Fig. 3.5 is the complete (Level-0) coordinator generated from the intermediate coordinator in Fig. 3.4. We can see from Fig. 3.5 that variable *g* can be bound to all the three possible values: *regular*, *plus*, and *premium*, since the CPN places related to these values (*p1* for *(gas, regular), p2* for *(gas, premium)* and *p4* for *(gas, plus)*) are connected to the CPN place *p3,* which represents *(gas, g)*. The connection is realized by binding transitions. For instance, binding transitions *t13* (*ft*) and *t31* (*bt*) connect *p1* and *p3*. Since transition Customer1_Pump is connected to *p3*, which represents the token *(gas, g)*, *g* can be bound to three different values - *regular*, *plus*, and *premium-,* and the Customer1 agent is entitled to obtain all three types of gas: *regular*, *plus*, and *premium*. In contrast, Customer2 can only obtain regular gas, because transition Customer2_Pump is connected to *p1*, which represent the token (gas, regular), containing no variable. Here *t13* can convey token *(gas, regular)* from *p1* to *p3*, so that value *regular* is bound to variable *g*. If *(gas, regular)* is not consumed by firing Customer2_Pump for any reason, it can be returned to *p1* through *t31*, so that Customer2 can consume *(gas, regular)* by firing transition *Customer2_Pump*.

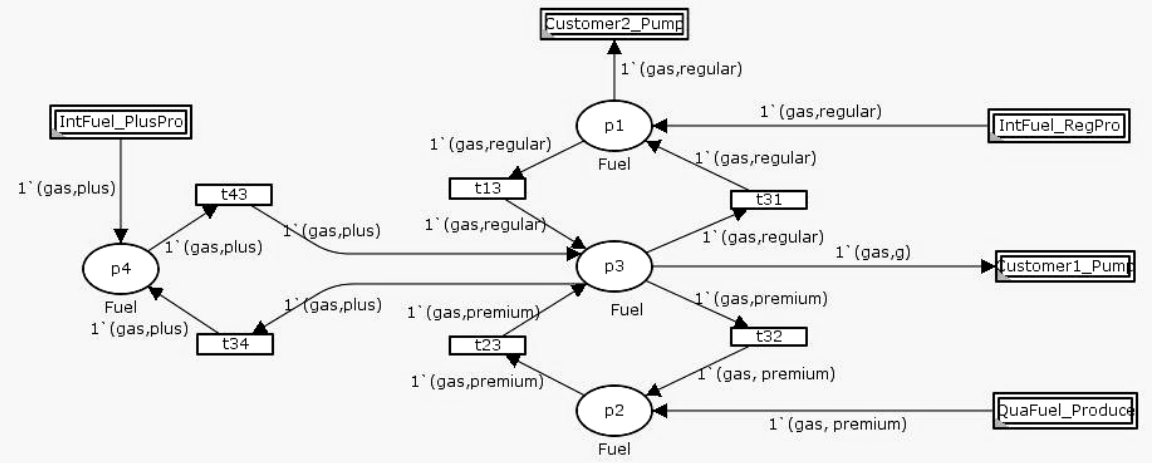


Fig. 3.5 Complete (Level-0) coordinator for the gas production/consumption MAS

3.2 The Level-1 Coordinator

Beyond the initial resource type matching, it is a common practice for an agent to only accept a subset of qualified resources for various reasons. Within this subset, an agent might decide at run-time which resource to use depending on the environment feedback. For example, a car renter may know he/she wants a mid-size car ahead of time, but he/she is willing to accept any available mid-size car at the rental location. In this case, mid-size car is the subset to be chosen from. Defining a subset of acceptable resources is a simple category of resource constraints that can be requested by local agents, and imposed by the coordinator. These types of resource constraints are called *resource matching constraints*. Processing resource matching constraints will evolve a Level-0 coordinator to Level-1. The new Level-1 coordinator mandates that only a subset of matching resource can be allocated, under the condition that the resource types have been matched between the requester agent and the supplier agent.

The resource matching constraints can be implemented by binding policies. By explicitly defining a set of binding policies as the resource matching constraints, we exclude those binding policies that are not explicitly defined (In contrast, a Level-0 coordinator includes all the possible binding policies.) The binding policies explicitly included in the resource matching constraints define the subset of values that a variable can be bound to, and therefore specify a subset of acceptable resource after type match. After the binding policies are integrated, the Level-0 coordinator evolves to level 1. Take the *production/consumption* MAS for example. The consumer may change its gas consumption policy due to a sudden budget cut-off. He can only afford regular and plus gas from the next month. Such a policy change can be implemented as a set of resource matching constraints s = {*((gas, regular), (car, g)), ((gas, plus), (car, g))}*. Fig. 3.6 shows the Level-1 coordinator, which is generated by integrating the set of binding policies *s*. Integrating *s* to the Level-1 coordinator prevents the consumer from obtaining premium gas.

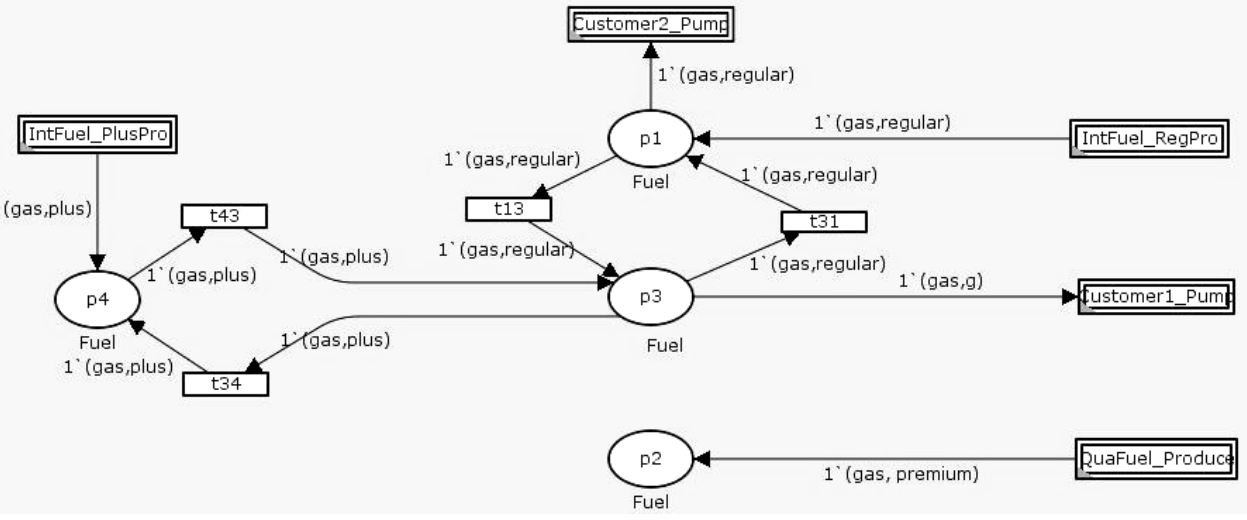


Fig. 3.6 Level-1 coordinator for the gas production/consumption MAS

As we can see from Fig. 3.6, binding transitions t23, t32 and the surrounding arcs of these two transitions are removed from Fig. 3.5 to generate the Level-1 coordinator shown in Fig. 3.6. Therefore, Customer1 can no longer obtain the premium gas. If all the possible binding policies are explicitly included in the resource matching

constraints, there exist no more variable binding restrictions beyond the resource type matching in the coordinator. In this case, the Level-0 coordinator is identical to the Level-1 coordinator. For example, if all three possible binding policies *((gas, regular), (gas, g)), ((gas, plus), (gas, g))* and *((gas, premium), (gas, g))* are included in the Level-1 coordinator, the Level-0 coordinator and Level-1 coordinator for the gas production/consumption MAS will be the same. In the rest of this paper, we use the Level-1 coordinator that is identical to the Level-0 coordinator.

To generate the Level-1 coordinator, we only need to remove those binding policies not explicitly specified in the resource matching constraints, since the Level-0 coordinator already includes all the possible binding policies. Because a binding policy can be removed by removing the two related binding transitions and the surrounding arcs to the binding transitions, the generation of Level-1 coordinator can be easily automated by the following algorithm. (See Appendix A for comments on the algorithm's correctness.)

Level-1 Coordinator Generation Algorithm

Input: A set of binding policies s = $\{b_i | b_i$ is a binding policy$\}$, A Level -0 coordinator C= (Σ, P, T, A, N, C, G, E, I)

Output: A Level -1 coordinator C

For each possible binding policy $b_i = (pre, post)$ not belonging to s

(Supposed place *Pre* represents resource *pre* in *C,* place *Post* represents resource *post* in C, *ft* is the forward biding transition for b_i, and *bt* is the backward biding transition for b_i)

Delete the two binding transitions *(ft, bt)* connecting place Pre and place Post.
Delete the four arcs (*Pre, ft), (ft, Post), (Post, bt)* and *(bt, Pre).*
End for
Return C.

3.3 The Level-2 Coordinator

Binding policies can only include or exclude specific subsets of resources a local agent can obtain/ release through the external environment, but it cannot specify the preference of resource when multiple matching resources are available. In order to specify the resource preference, we introduce a new concept called a *local policy*. Local policies are used to address the relationship between binding policies, so that the MAS designers can use local policies to address which resource an agent prefers when multiple choices are available. Integrating local policies into a Level-1 coordinator will generate a Level-2 coordinator.

Definition - *Local policy (LP)*: A local policy is an expression to represent the relationship between multiple binding policies of an agent.

In this paper, we propose two types of local policies: *Priority local policy* and *Proportional local policy*. A priority local policy is used to represent the preference of one resource over another when multiple resource units are available, while a proportional local policy is used to mandate the distribution of resource usage when multiple resource unites are available. Exploring more local policy genres is one of our future research topics. In the rest of this section, we elaborate both local policies.

Definition - *Priority local policy (Priority LP):* Supposed both *b1* and *b2* are binding policies, a priority local policy can be represented as b1 > b2. (Only if b1 is not available, b2 is valid).

We continue our gas production/consumption MAS to demonstrate the application of a priority local policy. Supposed that the consumers prefer to use regular gas, but will accept plus gas when no regular gas is available. This preference can be specified by a priority local policy *b1 > b2,* where *b1 = ((gas, regular), (gas, g)),* and *b2 = ((gas, plus), (gas, g)).*

Fig. 3.7 shows the Level -2 coordinator with the binding policy *b1 > b2*.

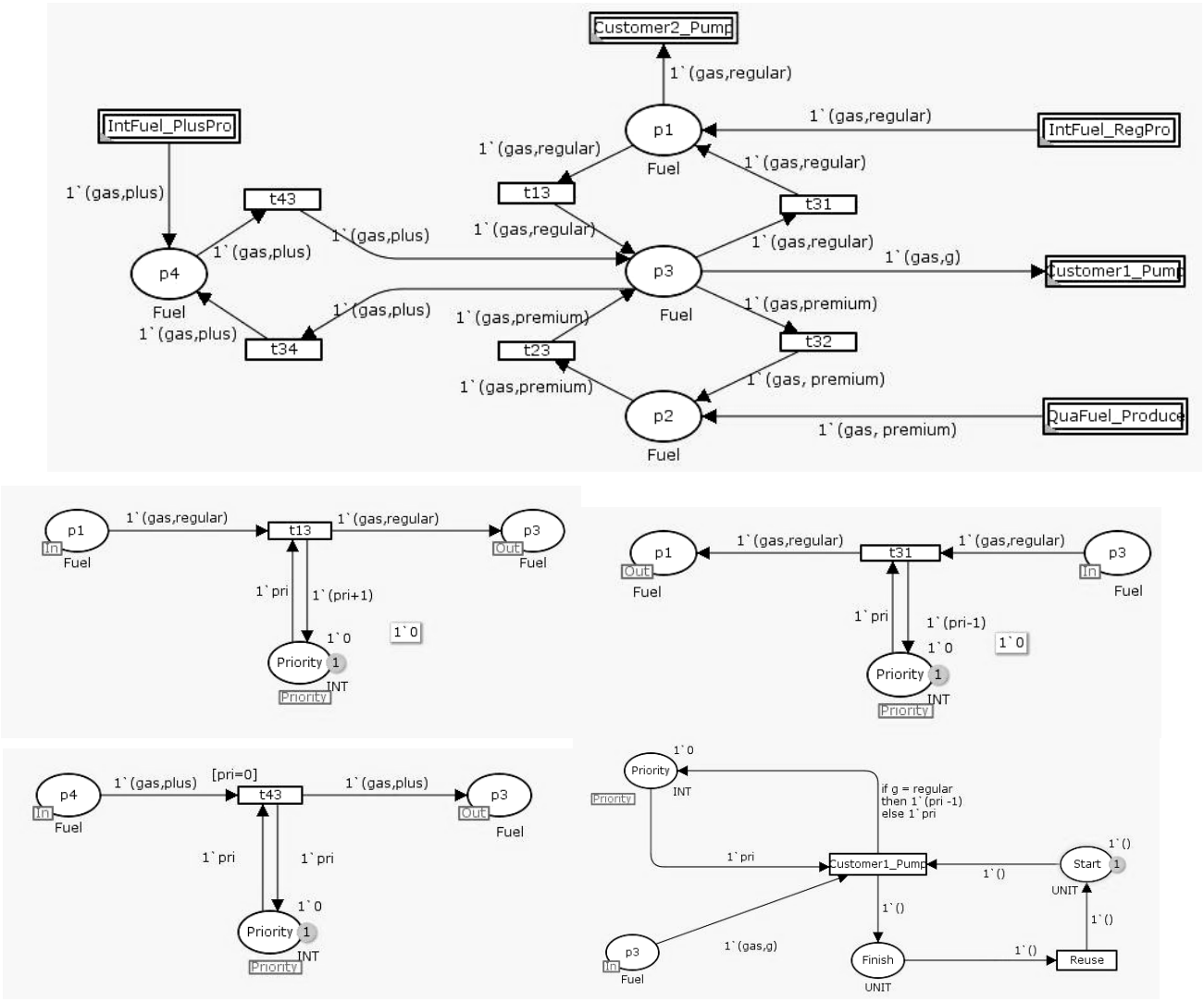


Fig. 3.7 Level-2 coordinator for the gas production/consumption MAS: with Priority LP *b1> b2*

b1 > b2 can be interpreted as: token *(gas, plus)* can be conveyed to place *p3* only when there is no token *(gas, regular)* in *p3*. Since binding transitions are the channel to convey resource tokens, *b1 > b2* can be imposed on relevant binding transitions without modifying other parts of the coordinator. Among the four binding transitions associated with *b1* and *b2*, *t13* and *t31* affect the availability of *(gas, regular)*, and further impact the binding of *(gas, plus)*. Therefore, *t13* and *t31* are relevant to *b1 > b2*. *t43* is also relevant, since it is the gateway to convey the resource *(gas, plus)* to p3, and is therefore under the influence of b1> b2. However, t34 is non-relevant, because it only provides the channel to retreat the (gas, plus) from p3, but has nothing to do with the binding priority.

As we can notice from Fig. 3.7, the Level-2 coordinator keeps all the elements in the Level-1 coordinator so that all the upper half of Level-2 coordinator is the same with the Level-1 coordinator (Fig. 3.5, since we use a Level-1 coordinator identical to a Level-0). The extra resource constraint is implemented through substituting transition *t43*, *t13*, and *t31* with a subpage respectively. Using substitution transitions, the model draws a clear boundary between existing elements of the Level-1 coordinator and the new elements added by the Level-2 coordinator. As a benefit, we separate the existing elements of Level-1 coordinator from extra resource constraints included in the model.

To impose the binding priority *b1 > b2*, a place called *priority* is added into the coordinator. The place *priority* holds an integer initial token *0.* The token *(gas, plus)* can be conveyed to place *p3* from *p4* only through its binding transition *t43* when *pri = 0*. (*pri = 0)* is defined as a guard condition associated with the binding transition *t43*. When a token *(gas, regular)* is conveyed to *(gas, g)* through its forward biding transition *t13,* the value of *pri is incremented.* When a token *(gas, regular)* is consumed or sent back to the place *p1*, we decrement the values of *pri*. This mechanism guarantees the *pri = 0* if and only if there is no *(gas, regular)* token on place p3.

Generally speaking, the implementation of priority local policy is always a series of the following operation: (1) add a control variable (*pri* in Fig 3.7) and a CPN place (*Priority* in Fig. 3.7) to the coordinator; (2) connect the CPN place to relevant binding transitions (transition *t43*, *t31*, and *t13* in Fig. 3.7) with arcs to monitor the available quantity of the preferred resource unit (*(gas, regular)* in Fig. 3.7*)*by the control variable; (3) use the control variable to control the availability of the non-preferred resource unit (*(gas, plus)* in Fig. 3.7*).* These operations can be automated by the following Priority LP Processing Algorithm. (See Appendix B for comments on the algorithm's correctness.)

Priority Local Policy (Priority LP) Processing Algorithm

Input: A priority local policy $b_1 > b_2$, A Level -1 coordinator C = (Σ, P, T, A, N, C, G, E, I)
Output: A Level -2coordinator C

(Suppose *ft* is the forward biding transition for b_i, and *bt* is the backward biding transition for b_i)

Add a control variable *v* and a CPN place *p* to the coordinator
// Add the following arcs to A:
Add r1 = (b_1.ft, p) to A, Add E(r1) = 1`(*v* + 1) to E;
Add r2 = (p, b_1.ft) to A, Add E(r2) = 1`*v* to E;
Add r3 = (b_1.bt, p) to A, Add E(r3) = 1`(*v* - 1) to E;
Add r4 = (p, b_1.bt) to A, Add E(r4) = 1`*v* to E;
Add r5 = (b_2.ft, p) to A, Add E(r5) = 1`*v* to E;
Add r6 = (p, b_2.ft) to A, Add E(r6) = 1`*v* to E;

// append the Guard condition
Add *v = 0* to G(b_2.ft).
Return C.

Proportional local policy is another type of local policy that regulates the ratio of resource consumptions among the agents.

Definition - *Proportional local policy (PLP):* Supposed both *b1* and *b2* are binding policies, a proportional local policy can be represented as *b1 / b2* = constant (Statistically, *b1* will occur constant times more frequently than *b2*).

We also use one example to illustrate the application of proportional binding in the gas production/consumption example. Customer2 has some coupons that allow purchase of premium gas at the price of regular gas once the customer has consumed regular gas for five times. The consumer can include the coupon as one of its local policies as: *b1/b2 =5*. Fig. 3.8 depicts the Level-2 coordinator for the gas production/consumption MAS with this proportional local policy. It is based on the Level-1 coordinator depicted in Fig. 3.6.

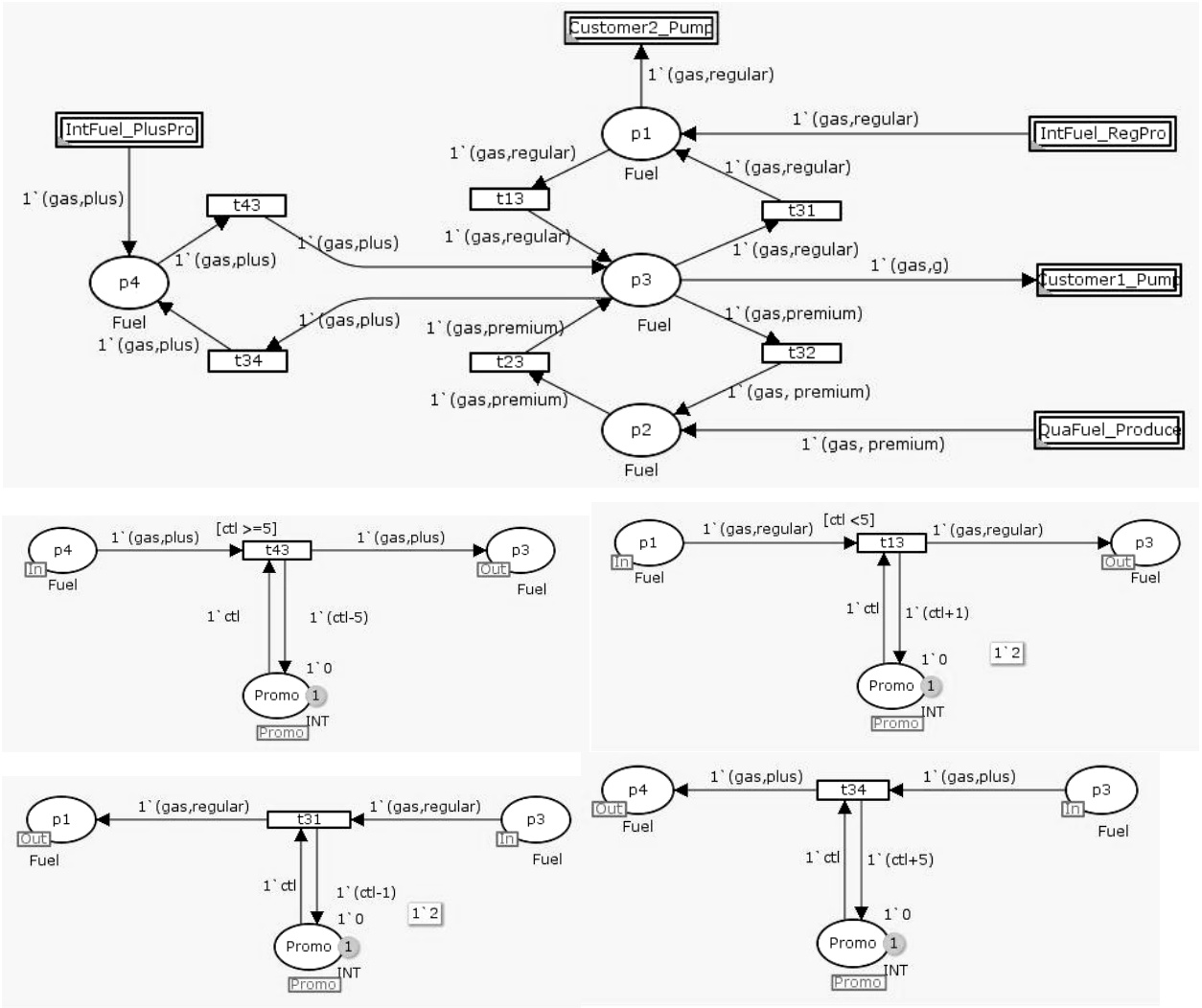


Fig. 3.8 Level-2 coordinator for the gas consumption/production MAS: with proportional LP *b1/ b2 =5*

The policy *b1/b2 =5* can be interpreted as: token *(gas, regular)* can be conveyed to place *p3* five times more often than token *(gas, plus)*. According to this interpretation, we need to count the number of times a binding occurs and use it as the control condition. Since binding transitions convey resource tokens, all the counting and controls can be imposed on relevant binding transitions without modifying other parts of the coordinator.

Similar to Fig. 3.7, the extra resource constraint is implemented through substituting transitions *t43*, *t34*, *t13*, and *t31* with a subpage. Therefore, the Level-2 coordinator in Fig. 3.8 keeps all the elements in the Level-1 coordinator; all the upper-half of the Level-2 coordinator is the same as the Level-1 coordinator (Fig. 3.6).

Fig. 3.8 shows an example for the proportional binding policy implementation. To impose the binding policy *b1| b2 = 5*, a CPN place called *promo* is added into the coordinator. The place *promo* holds an integer token *ctl,* who *is* initialized as *0.* Using the value of *ctl* as the guard condition*,* the coordinator controls the enablement of forwarding transitions for different resources*,* and consequently enforces the relevant resource consumption ratio*.* The token *(gas, plus)* can be conveyed to place *p3* from *p4* only through its binding transition *t43* when *ctl >= 5;* also, the token *(gas, regular)* can be conveyed to place *p3* from *p1* only when *ctl < 5. Ctl* is updated as follows:

When a token *(gas, regular)* is conveyed to *(gas*, *g)* through its forward biding transition *t13,* the value of *ctl* is incremented. When a token *(gas, regular)* is sent back to the place *p1* by *t31*, we decrement the values of *ctl*. When a token *(gas, plus)* is conveyed to *(gas, g)* through its forward biding transition *t13,* the value of *ctl* is incremented 5 times. When a token *(gas, regular)* is sent back to the place *p1*, we subtract the values of *ctl by 5*.

The implementation of a proportional local policy is always a series of the following operations: (1) add a control variable (*ctl* in Fig 3.8) and a CPN place (*Promo* in Fig. 3.8) to the coordinator; (2) connect the CPN place to relevant binding transitions (transition *t43*, *t34*, *t31*, and *t13* in Fig. 3.8) with arcs to monitor the ratio of binding transition firings by the control variable; (3) use the control variable to control the firing frequency of all binding transitions. The processing of a proportional binding policy can be realized by the following algorithm. (See Appendix C for comments on the algorithm's correctness.)

Proportional Local Policy (Proportional LP) Processing Algorithm

---

Input: A proportional local policy $b_1 \mid b_2 = c$ (c is a constant), A Level -1 coordinator C = (Σ, P, T, A, N, C, G, E, I)
Output: A Level -2coordinator C

(Supposed *ft* is the forward biding transition for b_i, and *bt* is the backward biding transition for b_i)

Add a control variable *v* and a CPN place *p* to the coordinator
// Add the following arcs to A:
Add r1 = (b_1.ft, p) to A, Add E(r1) = 1`(*v* + 1) to E;
Add r2 = (p, b_1.ft) to A, Add E(r2) = 1`*v* to E;
Add r3 = (b_1.bt, p) to A, Add E(r3) = 1`(*v* - 1) to E;
Add r4 = (p, b_1.bt) to A, Add E(r4) = 1`*v* to E;
Add r5 = (b_2.ft, p) to A, Add E(r5) = 1`*v* to E;
Add r6 = (p, b_2.ft) to A, Add E(r6) = 1`(*v* - *c)* to E;
Add r7 = (b_2.ft, p) to A, Add E(r5) = 1`*v* to E;
Add r8 = (p, b_2.ft) to A, Add E(r6) = 1`(*v* + *c)* to E;

// append the Guard condition
Add *v* >= *c* to G(b_2.ft).
Add *v* < *c* to G(b_1.ft).

Return C.

3.4 The Level-3 Coordinator

It is common that individual agent designers are only concerned about local interests, which may ignore the global interest of the MAS system. Therefore, to satisfy the vantage point from the global level, coordination policies that are independent of individual agents' interests are necessary. A typical example can be found in the investment banking industry, where a trader (modeled as an agent) is most concerned about personal gain and gives little thought to the wider strategy of the bank; so the bank has to enforce some global strategy that is independent of the interests of individual traders. To enforce the coordination control from the global level, we introduce a concept called *global policy*. As distinguished from local policies, global policies depict how resources are shared between agents, not within an agent.

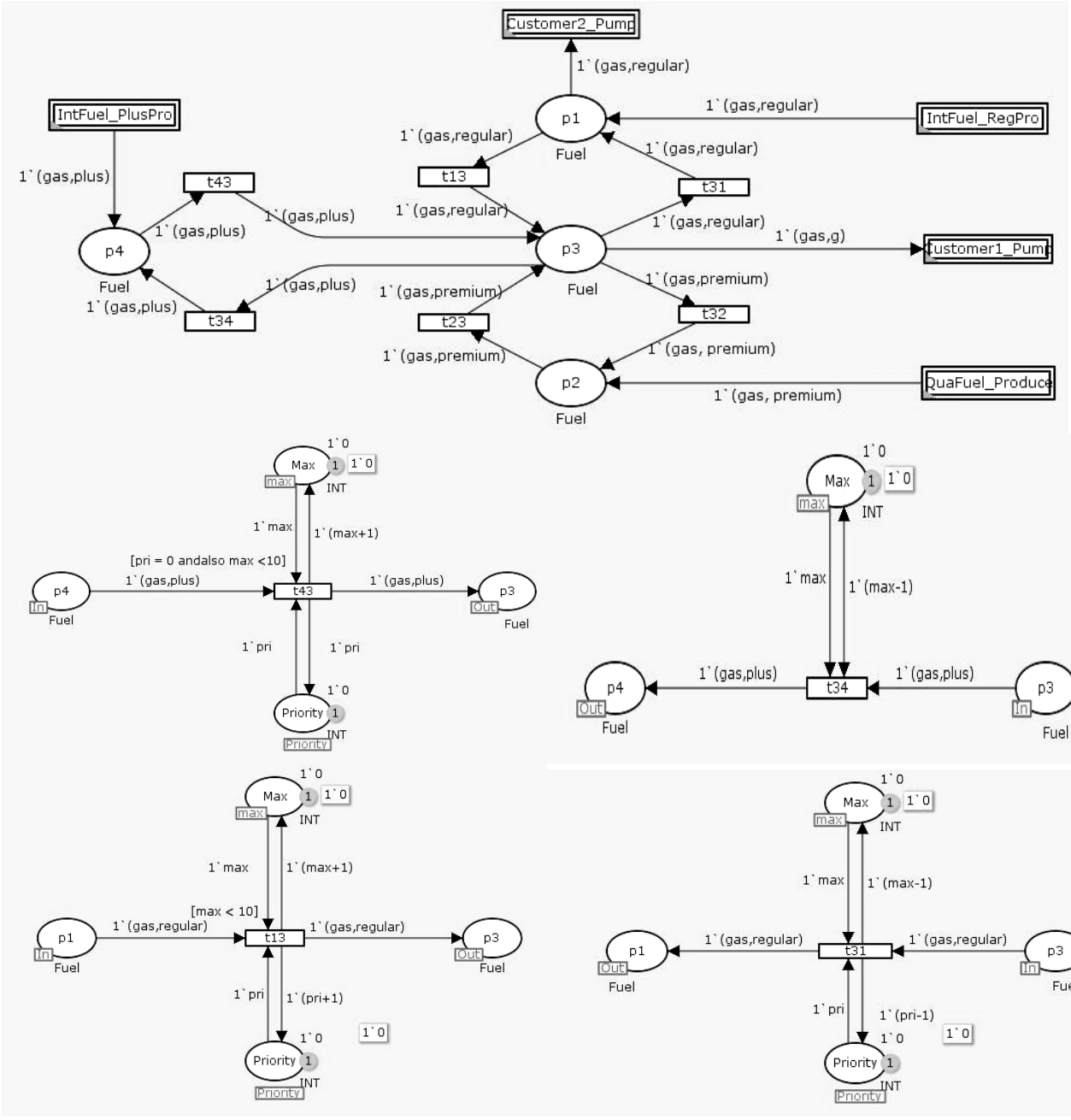
We have identified two types of global policies, namely *Universal global policy* and *Customized global policy*. A universal global policy imposes a certain restriction to a type of resource in the MAS, while a customized global policy outlines the resource coordination mechanism between agents. Exploring more formats of global policies is a topic for future research.

3.4.1 Universal Global Policy

We define a universal global policy as follows:

Definition - *Universal global policy*: A universal global policy takes the format $|res| < constant,$ where *res* is the resource name, and *constant* is an integer value.

For our gas production/consumption MAS, it is a common practice to limit the gas storage space, so that we can only store a limited quantity of fuel. This constraint can be enforced by a universal policy: $|(gas, g)| < 10$, which means that we cannot store over 10 gallon fuels at any moment (Assuming that the gas producer produces one gallon of fuel per time). As we have mentioned before, the global policy can be processed as an enhancement to the Level-2 coordinator, so that the resulting Level-3 coordinator preserves all the resource sharing constraints of a Level-2 coordinator. The Level-2 coordinator is enhanced by new elements appended to the existing substitution transitions.


Customer2_Pump
1`(gas,regular)
IntFuel_PlusPro
p1
Fuel
1`(gas,regular)
IntFuel_RegPro
1`(gas,regular)
1`(gas,regular)
1`(gas,plus)
t43
t13
t31
1`(gas,regular)
1`(gas,plus)
1`(gas,regular)
p4
Fuel
1`(gas,plus)
p3
Fuel
1`(gas,g)
Customer1_Pump
1`(gas,plus)
1`(gas,premium)
1`(gas,premium)
1`(gas,plus)
t34
t23
t32
1`(gas,premium)
1`(gas, premium)
p2
Fuel
QuaFuel_Produce
1`(gas, premium)
1`0
Max
1
1`0
max
INT
1`max
1`(max+1)
[pri = 0 andalso max <10]
p4
In
Fuel
1`(gas,plus)
t43
1`(gas,plus)
p3
Out
Fuel
1`pri
1`pri
1`0
Priority
1
INT
Priority
1`0
Max
1
1`0
max
INT
1`max
1`(max-1)
1`(gas,plus)
t34
1`(gas,plus)
p4
Out
Fuel
p3
In
Fuel
1`0
Max
1
1`0
max
INT
1`max
1`(max+1)
[max < 10]
p1
In
Fuel
1`(gas,regular)
t13
1`(gas,regular)
p3
Out
Fuel
1`pri
1`(pri+1)
1`0
1`0
Priority
1
INT
Priority
1`0
Max
1
1`0
max
INT
1`max
1`(max-1)
p1
Out
Fuel
1`(gas,regular)
t31
1`(gas,regular)
p3
In
Fuel
1`pri
1`(pri-1)
1`0
1`0
Priority
1
INT
Priority

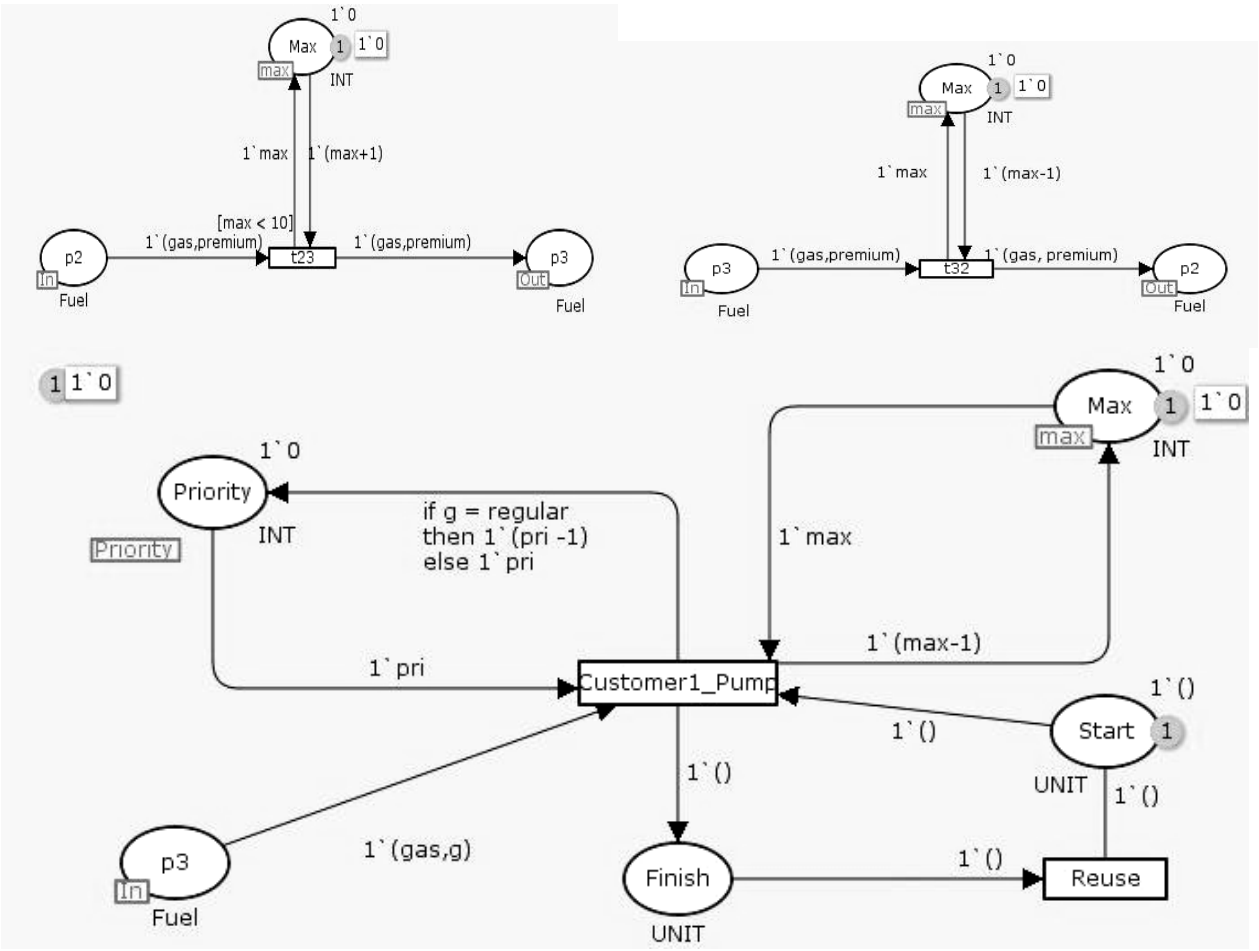


Fig. 3.9 Level-3 coordinator for the gas production/ consumption MAS: with universal global policy

Fig.3.9 illustrate how the universal global policy |*gas, g*| < *10* is implemented. In Fig. 3.9, a control value *max* is added to Level-2 coordinator to keep track of the number of available tokens -- *(gas, regular)* -- in place p3, and we append guard condition *[max < 10]* to transition *t43* and *t13* to ensure that the available gas will not exceed the capacity of the storage. As we can notice from Fig. 3.9, the Level-3 coordinator keeps all the elements in the Level-2 coordinator so that all the upper half of Level-3 coordinator is the same with the Level-1 and Level-2 coordinator (Fig. 3.6., Fig. 3.8).

Each universal global policy is implemented by a series of the following operations: (1) add a control variable (m*ax* in Fig 3.9) to denote the quantity of available resources, and a CPN place (*Max* in Fig. 3.9) to denote the storage of resource; (2) connect the CPN place to relevant transitions (transition *t43*, *t34,t31,t13, t23, t32 and Customer_Pump* in Fig. 3.9) with arcs to monitor the available quantity of the resource unit (*(gas, g)* in Fig. 3.9) by the control variable; (3) use the control variable to control the availability of the non-preferred resource unit (*(gas, plus)* in Fig. 3.9). These operations can be automated easily by the following algorithm. (See Appendix D for comments on the algorithm's correctness.)

Universal Global Policy Processing Algorithm

Input: A universal global policy $|p| < c$ (c is a constant), A Level -2 coordinator C = (Σ, P, T, A, N, C, G, E, I)
Output: A Level -3coordinator C
(Supposed resource unit p is modeled by Place P in C)

Add a control variable *v* and a CPN place *max* to the coordinator

For each *ft* connected to *P*
Add r1 = (*ft*, max) to A, Add E(r1) = 1`(*v* + 1) to E;
Add r2 = (max, *ft*) to A, Add E(r2) = 1`*v* to E;
Add *max* < *c* to G(*ft*).
End For

For each *bt* connected to P
Add r1 = (*bt*, max) to A, Add E(r1) = 1`(*v* - 1) to E;
Add r2 = (max, *bt*) to A, Add E(r2) = 1`*v* to E;
End For

For any of the rest transition *t* (those transitions use consume the resource p)
Add r1 = (*t*, max) to A, Add E(r1) = 1`(*v* -1) to E;
Add r2 = (max, *t*) to A, Add E(r2) = 1`*v* to E;
End For

Return C.

3.4.2 Customized Global Policy

The resource coordination constraint between agents is another type of resource constraint. It cannot be proposed by individual agents, since designing such constraints usually requires the information from multiple agents. However, this constraint genre is useful for the coordination from the global perspective. We extend our gas production/consumption MAS as an example. To improve the production quality, Quality Fuel Inc. adopts a complicated and time consuming production procedure. Not surprisingly, Quality Fuel Inc. produces fuels much slower than International Fuel Inc. In order to avoid the scarcity of premium gas, the MAS has to impose the following resource sharing constraint: Customer1 can use the gas from Quality Fuel Inc. only if there is no gas available from International Fuel Inc. Such a design constraint cannot be proposed by any local agent, since none of them can be aware of the disparity of production efficiency from the local information available to them. As a result, such a constraint has to be proposed in the global level by the coordinator designer, since the coordinator designer has access to the resource coordination requirements from multiple agents.

A customized global policy (CGP) is the type of global policies used by the coordinator designer to enforce resource coordination constraints between agents. Similar to local policies, a customized global policy is used to address a *priority* relationship or *proportional* relationship. But, as local policies applies to the resource with an agent, customized global policies address the resource coordination relationship among agents. Similar to the local policies, we have identified two types of customized global policies: *priority CGP* (to address the priority relationship between two agents) and *proportional CGP* (to address the proportional relationship between two

agents). In fact, we can treat a customized global policy as the global version of local policy, where the binding policies in a local policy are now replaced by agent names.

Definition - *Priority customized global policy (Priority CGP)*: For any two agents *AP1* and *AP2*, *AP1 > AP2* is a priority customized global policy.

Definition - *Proportional customized global policy (Proportional CGP)*: For any two agents *AP1* and *AP2*, AP1|AP2 = *constant* is a proportional customized global policy, where *constant* is an integer.

The constraint needed previously – *Customer1 can use the gas from Quality Fuel Inc. only if there is no gas available from International Fuel Inc.* – can be represented by the following Priority CGP: *IntFuel > QualityFuel*.

A CGP *AP1 > AP2* can be interpreted as follows. For any binding policy *b1* from the local plan of *AP1*, and any binding policy b2 from the local plan of *AP2*, *b1 > b2*. For instance, supposed *AP1* has binding policies *{b1, b2}* and *AP2* has binding policies *{b3, b4}*, a priority CGP *AP1 > AP2* can be interpreted as follows: *b1 > b3*, *b1 > b4, b2 > b3, and b2 > b4.* Similarly*, a proportional CGP AP1 | AP2 = constant* can be interpreted as follows: *b1 |b3 = constant*, *b1 |b4 = constant, b2 |b3 = constant, and b2 |b4 = constant.*

Based on the interpretation above, the implementation of a CGP can be derived from the implementation of a LP, and automated with a combination of LP automation algorithms proposed in this paper. Since the implementation of a CGP is similar to a corresponding LP, we omit it here to avoid redundancy.

4. Property Validation by Simulation

To further analyze the impacts of different coordinators on the gas production/consumption MAS model, we use the simulation capacities provided by CPN Tools. For the convenience of the property verification, we install a resource generator for each local plan to describe the continuous resource production and consumption over a period of time. The resource generator is a Timed Petri net model, representing the generation of resource at a specific frequency. More details about Timed Petri net can be found at [CPN06]. While the installation of the resource generator does modify the local plan of the agents, as a benefit of our component-based MAS modeling method, this modification is transparent to the coordinator. As a result, no coordinator needs to be modified. We simply concatenate the modified local plans with an existing coordinator to generate a new global plan, and run the experiments upon this global plan to verify various properties for the gas production/consumption MAS that we have modeled.

For example, Fig. 4.1 is a resource generator installed for the local plan of the Quality Fuel Agent. In Fig. 4.1, *()@+6* means generating a token unit *()* after 6 seconds. (Supposedly the number 6 in *()@+6* denotes 6 seconds here). The generated token *() d*rives the transition *QualityFuel_Produce* to generate a gas token *(gas, premium)*. Therefore, a unit of premium gas will be generated every 6 seconds in this model. The resource generators for other local plans are similar, and we omit them here for the concern of space. These resource generators generate a unit of premium gas every 6 seconds, a unit of plus gas every 4 seconds, and a unit of regular gas every 2 seconds. Also, we assume that the Customer1 agent needs to consume a gallon of gas every 2 seconds, and modify the local plan accordingly.

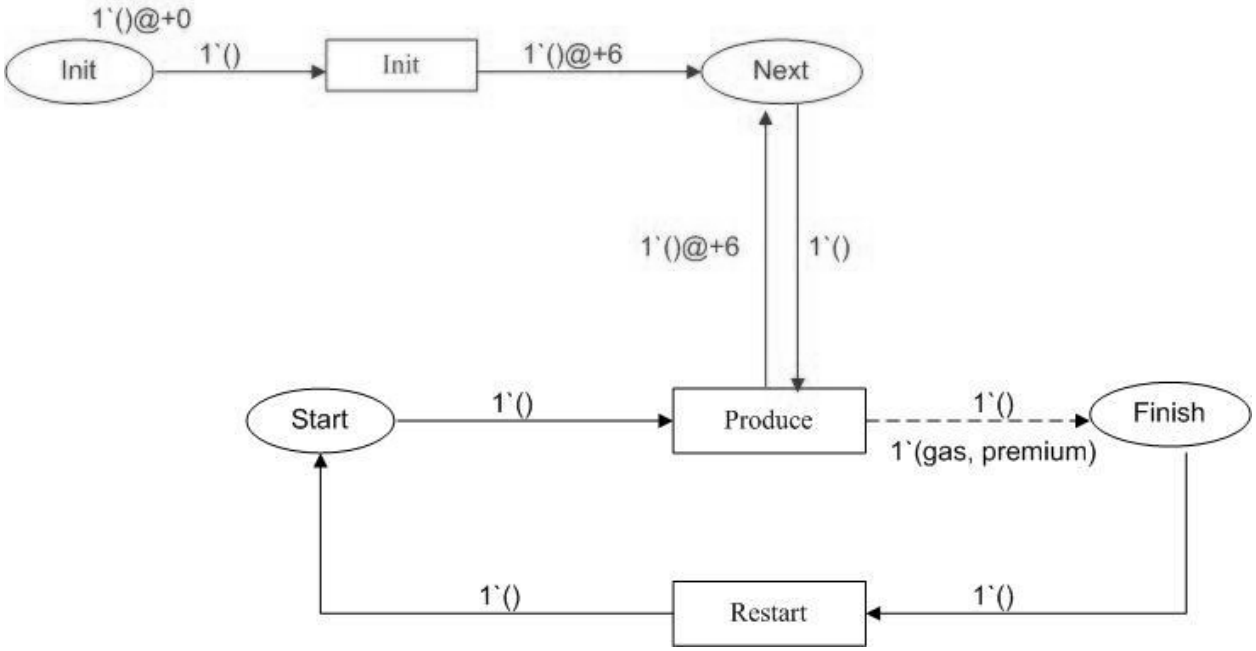


Fig. 4.1 Plus gas generator – produce a gallon of gas every 4 seconds

After the resource generators have been installed, we concatenate the local plans with the two different coordinators to build two global plans, and then ran a series of simulation to verify model properties.

Coordinator used to build the global plan	Regular		Plus		Premium		Simulated Execution Time(seconds)
	Consumed	Left	Consumed	Left	Consumed	Left	
Level-0	956	809	479	403	329	259	3530
	955	809	482	400	327	261	3530
	954	811	493	389	317	271	3530
Level-1	1261	614	613	324	0	625	3750
	1269	605	606	331	0	625	3750
	1249	625	626	611	0	625	3750

Fig. 4.2 Gas consumption comparison for Level-0 and Level-1 coordinators

Fig 4.2 compares the results for the MAS models using the Level-0 coordinator (Fig. 3.5) and the Level-1 coordinator (Fig. 3.6). The simulation was run three times for each coordinator, where each simulation composes of 10,000 transition firings. Fig. 4.2 records the number of tokens consumed during the simulation. As we can identify easily from Fig. 4.2, no premium gas was consumed when a Level-1 coordinator is used. This finding is consistent with our design of the Level-1 coordinator, where we disable the binding of premium gas to the

consumer. In contrast, for the Level-0 coordinator, all three types of gases are consumed in the rough proportion of their production speed.

We also perform simulation experiments on the Level-3 coordinator in Fig. 3.9. The Level-3 coordinator has a local priority policy that requires that plus gas be consumed only if there is no regular gas available. It also has a universal global policy that requires no more than 10 gallons of gas were stored any time. The gas production rate remains the same as the previous experiment. (2 seconds per regular gas unit, 4 seconds per plus gas unit, and 6 seconds per premium gas unit.)

Coordinator	Regular Consumed	Plus Consumed	Premium Consumed	Simulated Execution Time(seconds)
Level-3	1741	1	863	5212
	1739	1	866	5212
	1740	1	864	5212

Fig. 4.3 Gas consumption with a Level-3 coordinator

Fig.4.3 shows the result. As we can see from Fig. 4.3, the local priority policy disables the consumption of plus gas when regular gas is available. Although we can still see one unit of plus gas consumed, this finding is consistent with our local policy. The plus gas unit was consumed for the first time at time clock = 4(the fourth second after initial time). When time clock = 4, there is no regular gas since the regular gas generated at time 2 has been consumed. A regular gas unit and a plus gas unit will both be produced at time clock = 4. Whether the regular gas unit or the plus gas unit is generated first is controlled by the simulator randomly, therefore, it is possible that a plus gas unit be generated first when there is no regular gas, and then get consumed before a regular gas unit has been generated.

This simulation trace also confirmed that there are no more than 10 tokens in place *Max of Fig. 3.9* anytime during the simulation. This finding confirms that the universal global policy – no more than 10 gallons of gas were stored any time – is effective.

5. Coordinator Reduction for Homogeneous Agents

As was shown in Section 3, all ERT entries are processed for the level-0 coordinator generation. Such a design provides a universal interface for coordinator generation, independent of the characteristics of any agent in the MAS. Nevertheless, it is expected that some agents in a MAS will exhibit similar characteristics with regard to their interaction needs. By exploiting this similarity among agents, we can create a more concise coordinator, based on the idea of model reduction, without impeding the component's coordination capacity. This section introduces the concept of *agent homogeneity*, and presents an approach for creating compact coordinator models for systems composed of homogeneous agents.

For the purpose of illustrating the idea of homogeneous agents and coordinator reduction, let us extend the common example in Section 3 by adding a new fuel provider called Efficient Fuel Inc. (denoted as *EffFuel*) and a new customer called *Customer3*. As was the case for the agent International Fuel Inc., Efficient Fuel Inc. also produces regular gas and plus gas, but in a sequential way. The new agent Customer3 has the same local plan as Customer 2; thus it exclusively consumes regular gas. Fig. 5.1 shows the updated local plans for all the agents in the example MAS. Fig 5.2 shows the updated ERT table used for level-0 coordinator generation.

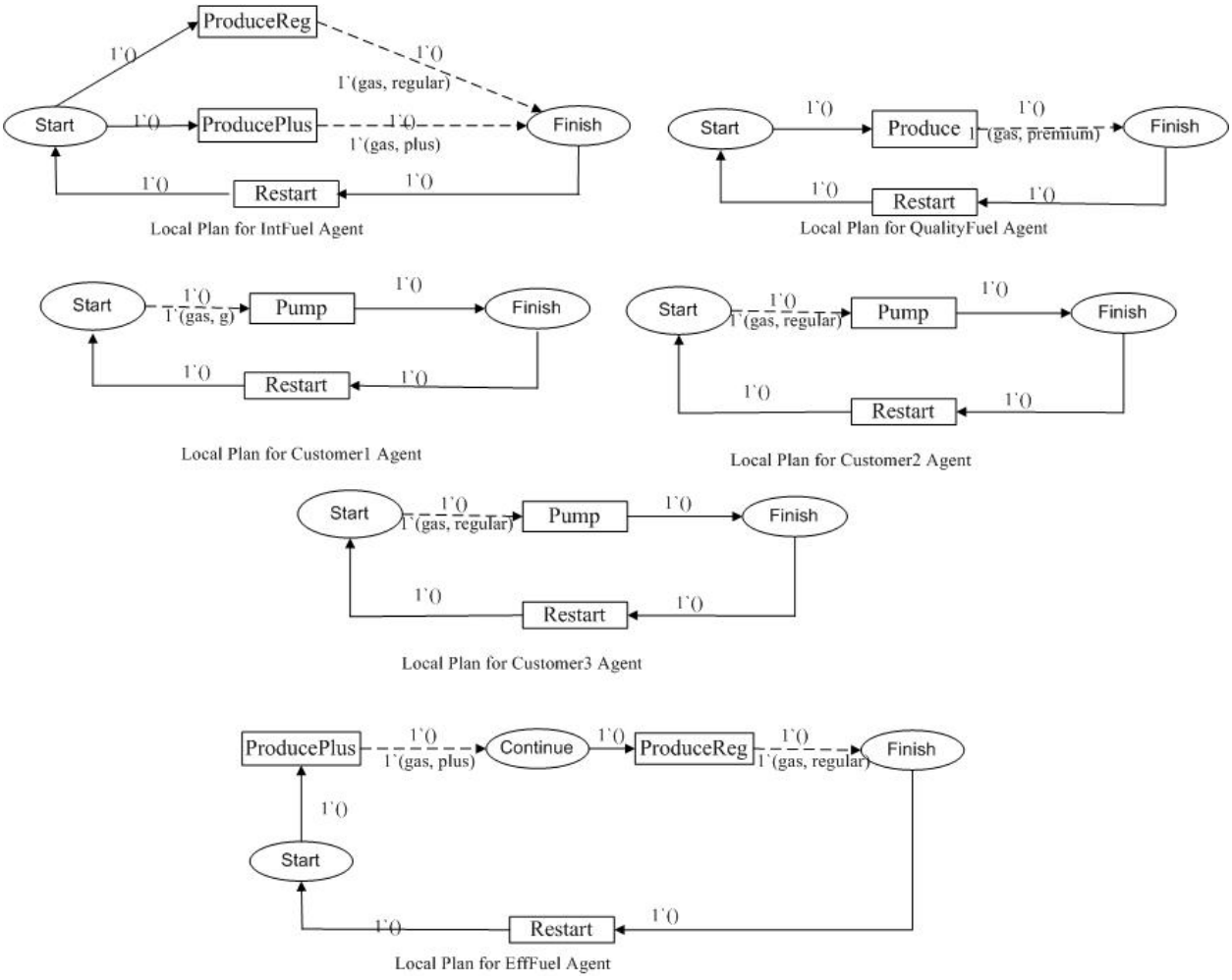


Fig. 5.1 Extended local plans for the gas production/consumption MAS

ERT Table	Entries
Customer1.ERT	(Customer1.Pump, request, 1`(gas, g))
Customer2.ERT	(Customer2.Pump, request, 1`(gas, regular))
Customer3.ERT	(Customer3.Pump, request, 1`(gas, regular))
IntFuel.ERT	*a* = (IntFuel.RegPro, release, 1`(gas, regular))
IntFuel.ERT	*b* = (IntFuel.PlusPro, release, 1`(gas, plus))
QualityFuel.ERT	(QualityFuel.Produce, release, 1`(gas, premium))
EffFuel.ERT	*c* = (EffFuel.RegPro, release, 1`(gas, regular))
EffFuel.ERT	*d* = (EffFuel.PlusPro, release, 1`(gas, plus))

Fig. 5.2 ERT for the extended gas production/consumption MAS

5.1 Agent Homogeneity

This section defines a set of concepts related to agent homogeneity. The concepts are applied to describe different degrees of similarity between agents, and provide the basis for coordinator reduction.

Definition – Two ERT entries $(t_1, type_1, PI_1)$ *and* $(t_2, type_2, PI_2)$ are considered *homogeneous ERT entries* iff $type_1 == type_2$ and $PI_1 == PI_2$.

In Fig. 5.2, ERT entries *(Customer2.Pump, request, 1`(gas, regular) )* and *(Customer3.Pump, request, 1`(gas, regular))* are homogeneous.

Definition – Two agents A_1 and A_2 are considered *semi-homogeneous agents* iff (1) they have the same number of ERT entries; (2) There exists a one-to-one mapping f, $\{f(e_i) = e_j \mid e_i \in A_1.ERT \text{ and } e_j \in A_2.ERT\}$, between the ERT entries of *A1 and A2* so that e_i and $f(e_i)$ are homogeneous.

In Fig.5.2, agents *EffFuel* and *IntFuel* are semi-homogeneous because they both have two ERT entries (i.e., *a* and *b* for *IntFuel* and *c* and *d* for *EffFuel*); and there exists a one-to-one mapping f, $\{f(a) = c$ and $f(c) = d\}$, where *a* and *c* are homogeneous entries and *b* and *d* are homogeneous entries.

Definition – Two agents are considered *homogeneous agents* if they have the same local plan.

According to Fig. 5.1, *Customer 2* and *Customer3* are obviously homogeneous agents.

Theorem: If two agents are homogeneous, they are also semi-homogeneous. The proof for this theorem is trivial: For any two homogeneous agents, they have identical local plans, so (1) they have the same potential arcs and inscriptions, implying that they have the same number of ERT entries; and (2) There exists a one-to-one mapping f, where $f(a) = a$. ERT entry *a* is homogeneous to itself.

However, the opposite proposition does not hold. *EffFuel* and *IntFuel* provide a simple counter-example. These two agents are semi-homogeneous, but not homogeneous, because their local plans are not identical.

Definition – If two agents are semi-homogenous agents, but not homogeneous agents, they are considered *pure semi-homogeneous agents*.

Obviously, *EffFuel* an *IntFuel* are pure semi-homogeneous agents.

5.2 Coordinator Reduction

The level-0 coordinator associated with the ERT table of Fig. 5.2 is shown in Fig 5.3. Fig 5.3 exhibits some symmetry because homogeneous ERT entries from different agents are processed. For example, *Customer3_Pump* and *Customer2_Pump* come from two homogeneous ERT entries, so they produce some symmetry for the coordinator. Similar symmetries also exist between transition pairs *(IntFuel_PlusPro, EffFuel_PlusPro)* and *(IntFuel_RegPro, EffFuel_RegPro)*.

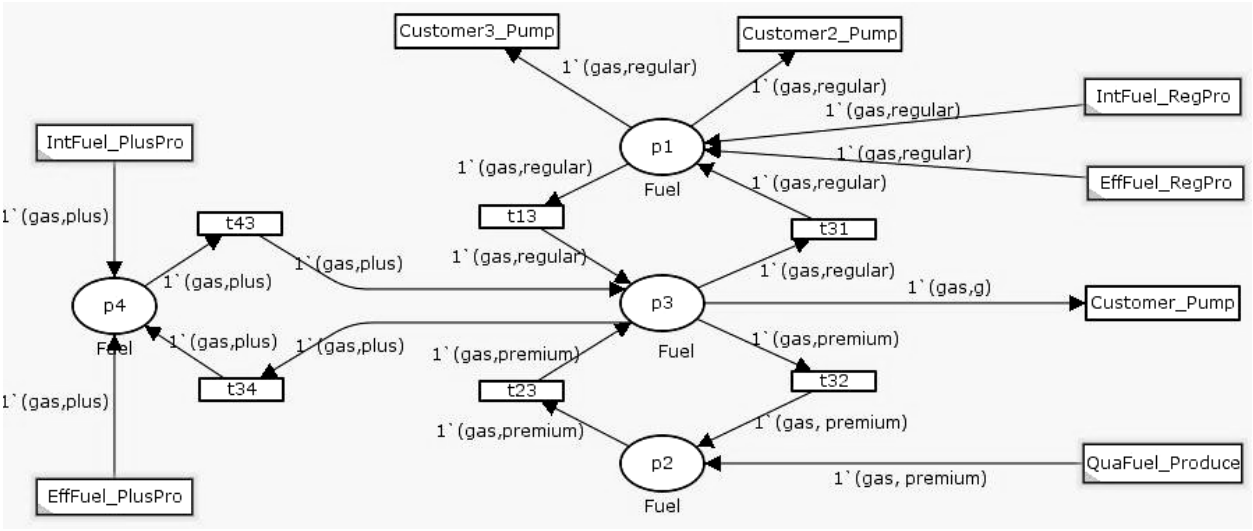


Fig. 5.3 Original level-0 coordinator generated by for the extended gas production/consumption MAS

It is possible to eliminate such redundant elements in the coordinator by merging ERT entries from semi-homogeneous agents (Reminder: all the homogeneous agents are also semi-homogeneous). As a result, the ERTs of Customer2 and Customer3 are merged into CustomerX, and the ERTs of EffFuel and IntFuel are merged into Fuel.ERT. Fig. 5.4 shows the new reduced ERT table.

ERT Table	Entries
Customer1.ERT	(Customer1.Pump, request, 1`(gas, g))
CustomerX.ERT	(Customer2.Pump, request, 1`(gas, regular))
Fuel.ERT	(RegPro, release, 1`(gas, regular))
Fuel.ERT	(PlusPro, release, 1`(gas, plus))
QualityFuel.ERT	(QualityFuel.Produce, release, 1`(gas, premium))

Fig. 5.4 The reduced ERT table for the extended gas production/consumption MAS

By processing the reduced ERT table using the original level-0 coordinator generation algorithm, a more concise level-0 coordinator can be produced and is shown in Fig. 5.5. In Fig. 5.5, transition *RegPro* is a transition produced by processing ERT entry *(RegPro, release, 1`(gas, regular))* from Fig. 5.4, and it can be viewed as a result

of merging transitions *IntFuel_RegPro* and *EffFuel_RegPro* from Fig. 5.3. Also, transition *PlusPro* and *CustomerX_Pump* are generated in a similar manner.

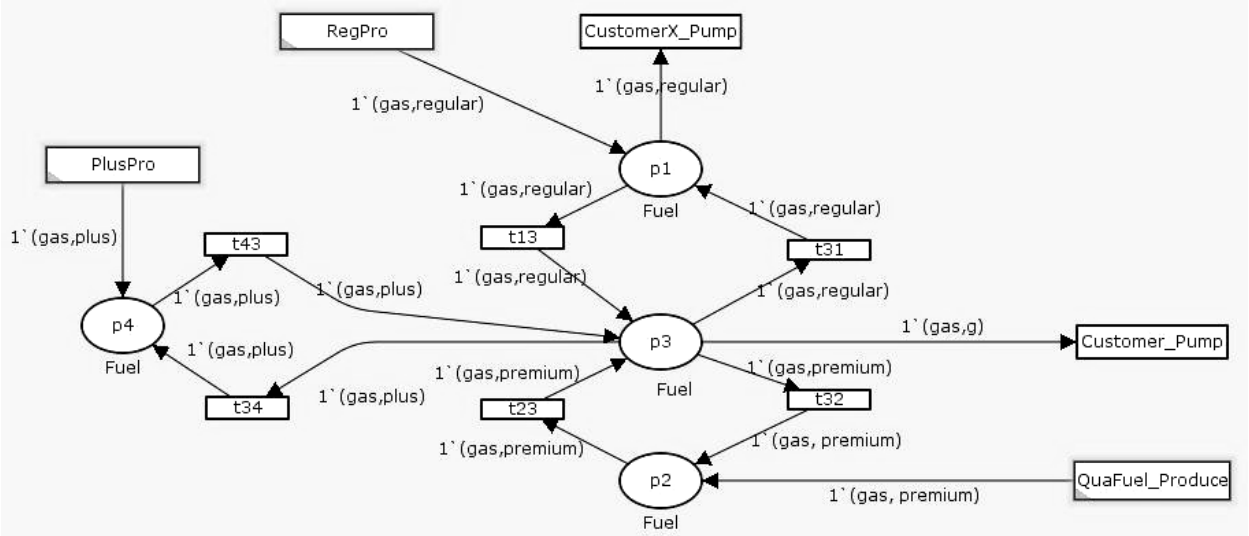


Fig. 5.5 The reduced level-0 coordinator generated by processing the reduced ERT table in Fig. 5.4

The reduced level-0 coordinator captures the coordination mechanism in the MAS, but it does not distinguish between semi-homogeneous agents. For example, the coordinator cannot tell if *CustomerX_Pump* is generated for one agent, *CustomerX*, or a set of semi-homogeneous agents. To address this problem, we introduce *agent tokens* and *agent token generators*. A similar idea can be found in [DC06], where an agent can be represented as a token inside an agent system. However in that approach the adopted architecture includes multiple agent systems, and the concern is with modeling agents that must migrate from one system to another to interact. In contrast, our approach is aimed at one agent system, and our concept of an agent token is used in the coordinator only for the special purpose of distinguishing semi-homogeneous agents, not for representing an arbitrary agent. By supplementing the reduced level-0 coordinator with agent tokens and agent token generators, we can distinguish coordinators generated from non-homogeneous agents and those generated from a set of homogeneous agents.

The coordinator from Fig. 5.5 is modified as follows: for each transition generated by processing a merged ERT entry, we add an agent token generator and a CPN place to hold the generated agent tokens. The agent tokens can then distinguish which agent is actually performing the associated actions, because the agent token now is required to enable a transition, and this agent token represents the performer of the action. Fig. 5.6 shows the reduced level-0 coordinators with agent tokens and agent token generators. In Fig. 5.6, transition *eIncGen* and *eCusGen* are agent token generators. The former generates agent tokens *IntFuel* and *EffFuel*, whereas the latter generates agent tokens *Customer2* and *Customer3*. Place *Customer* can hold token *customer2* and *customer3*, both of which belong to color set CX. If token *customer2* is generated by token generator *eCusGen* and deposited into place *Customer* to enable transition *CustomerX_Pump*, then agent *Customer2* is the action performer.

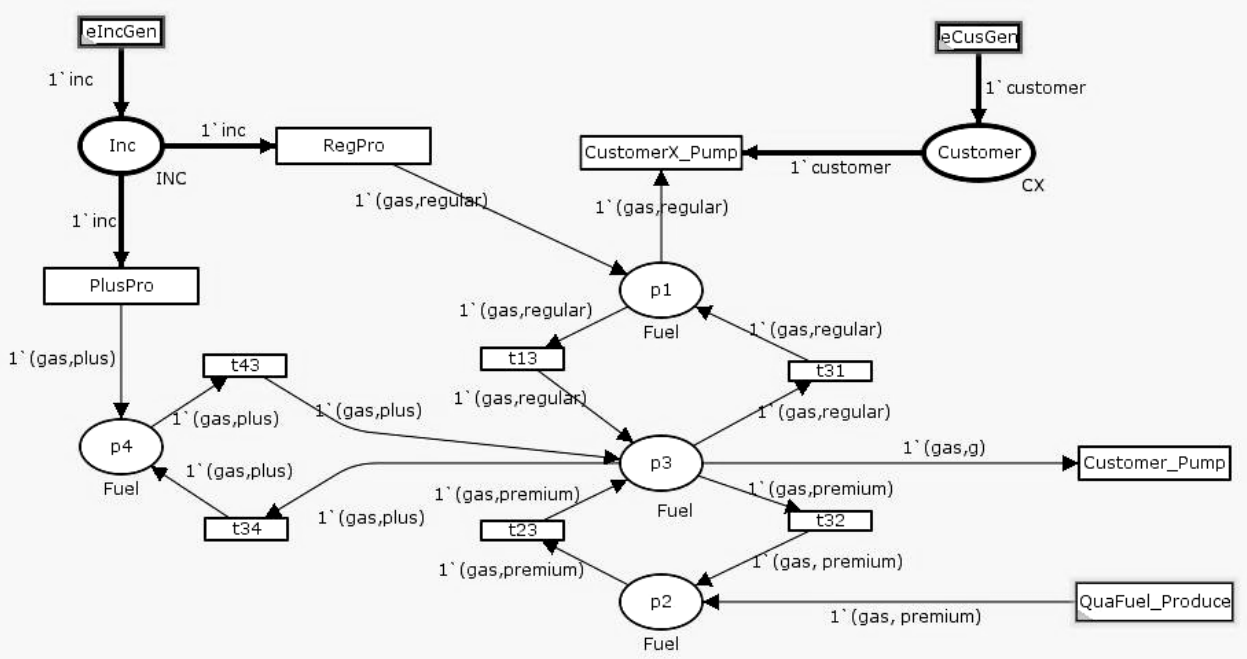


Fig. 5.6 The reduced level-0 coordinator with agent tokens and agent token generators

5.3 Discussion

Reducing the coordinator makes the coordinator component more concise, however, it may impact the following two actions: (1) global plan generation; and (2) achieving flexibility in coordination.

Semi-homogeneous agents and homogeneous agents are treated in the same way during coordinator generation. But, when it comes to the global plan generation, they should be processed differently. The only difference between homogeneous agents is the disparity of agent names. Therefore, it is enough to just use agent tokens to distinguish them, even in the global plan level. For a set of homogeneous agents, only one local plan needs to be integrated in the global plan. Semi-homogeneous agents represent a different case, since they exhibit different local behaviors in their local plans. As a result, ERT entries for pure semi-homogeneous agents have to be unfolded in the global plan generation process so that all the local plans from pure semi-homogeneous agents can be integrated in the global plan.

Another question is the impact of coordinator reduction on the processing of resource constraints to achieve flexibility in coordination. Because coordinator reduction makes use of agent homogeneity by merging homogeneous agents, this may affect the processing of those coordination inputs concerned with specific semi-homogeneous agents. For example, *priority customized global policy* $A_1 > A_2$, will be difficult to achieve by processing the reduced level-0 coordinator, if A_1 and A_2 are at least semi-homogeneous. The reason is that the two semi-homogeneous agents will have been merged in the coordinator level.

6. Conclusion and Future Research

We proposed a design method that focuses on creating flexible coordinator components for modeling multi-agent systems. The method defines four levels of coordination and an associated four-step coordinator

generation process that allows for the design of coordinators with increasing capabilities for handling the complexity associated with resource coordination. Colored Petri net-based simulation is used to analyze various properties that derive from different coordinators. For homogeneous agents, the coordinator can be reduced.

There are a number of areas for future research. First, it will be useful to study the integration of agent reasoning mechanisms into our modeling approach to provide more comprehensive agent models. This work will integrate the type of architecture-level modeling discussed here with specific agent-based capabilities found in the agent design literature. Second, just as the field of software engineering generalized specific design elements into the form of design patterns, it might be possible to define some net-based patterns of coordination to foster reuse in the process of agent modeling. A third area for future research is to focus on both simulation and model checking analysis, with the goal of providing support tools that simplify analysis for agent designers by hiding lower-level Petri net specific details. For multi-agent systems, as with any system of concurrent components, there is the inherent state-space explosion problem that challenges state-space analysis such as model checking. Thus, system scalability becomes an issue for that type of analysis, and it is appropriate, and probably necessary, to adapt techniques from other domains for mitigating the state-space explosion problem. Fourth, there is a need to bridge the gap between modeling and implementation, i.e., creating guidelines or rules for synthesis of runnable agents from the abstract modeling components. Finally, investigating the impact of coordinator reduction upon global plan generation and coordination input processing is also an important topic for further study.

References

[BM96] B. Tsvetovatyy and M. Gini, "Towards a Virtual Marketplace," *Proceeding of the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology*, 1996.

[CPN06] CPN ML Language, From CPN Tools (Version 2.0) help file, 2006.

[CF04] G.Cabri, L.Ferrari and L.Leonardi, "Agent Role-based Collaboration and Coordinations: a Survey about Existing Approaches," *Systems, Man and Cybernetics, 2004 IEEE International Conference*, 10-13 Oct. 2004.

[CG98] L. Cardelli and D. Gordon, "Mobile Ambients," *Foundations of Software Science and Computational Structures, Lecture Notes in Computer Science,* No. 1378, Springer-Verlag, Berlin, pp. 140-155, 1998.

[CL00] G. Cabri, L. Leonardi and F. Zambonelli, "MARS: a Programmable Coordination Architecture for Mobile Agents," *IEEE Internet Computing*, Vol.4, No.4, pp.26-35, Jul.-Aug., 2000.

[DC06] J. Ding, P. Clarke, D. Xu, X. He and Y. Deng, "A Formal Model-based Approach for Developing an Interoperable Mobile Agent System," *The Multi-agent and Grid Systems – An International Journal,* No. 4, Vol. 2, 2006.

[FI02] FIPA Contract Net Interaction Protocol Specification, from http://www.fipa.org/specs/fipa00029/, 2002.

[FT00] D. Fitoussi and M. Tennenholtz, "Choosing Social Laws for Multi-Agent Systems: Minimality and Simplicity," *Artificial Intelligence*, Vol. 119, Issues 1-2, pp. 61-101, May 2000.

[GH97] S. Green, L. Hurst, B. Nangle, P. Cunningham, F. Somers and R. Evans, "Software Agents: a Review," Intelligent Agent Group (IAG) report TCD-CS-1997-06, Trinity College Dublin, May 1997.

[HB04] C. Hanachi and C. Blanc, "Protocol Moderators as Active Middle-Agents in Multi-Agent Systems," *Autonomous Agents and Multi-Agent System,* Vol. 8, No. 2, pp. 131-164, 2004.

[HM04] R. Herrera and E. Mellado, "Modular and Hierarchical Modeling of Interactive Mobile Agents," *Systems, Man and Cybernetics, 2004 IEEE International Conference*, Vol. 2, pp. 1740 - 1745, 10-13 Oct. 2004.

[J96] N. Jennings, "Coordination Techniques for Distributed Artificial Intelligence," *Foundation of Distributed Artificial Intelligence, Sixth-Generation Computer Technology Series*, G. O`Hare and N. Jennings, Eds., New York: Wiley, pp. 187-210, 1996.

[J97] K. Jensen, "*Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use*," Springer Verlag, 2nd corrected printing, 1997.

[JS98] N. Jennings, K. Sycara and M. Wooldridge, "A Roadmap of Agent Research and Development," *Autonomous Agents and Multi-Agent Systems Journal*, Vol. 1, Issue 1, pp. 7-38, 1998.

[KG97] D. Kinny and M. Georgeff, "Modeling and Design of Multi-Agent Systems," *Proceeding of the 4th International Workshop on Agent Theories, Architectures, and Language* (ATAL-97), pp. 1-20, 1997.

[KG96] D. Kinny, M. Georgeff and A. Rao," A Methodology and Modeling Technique for Systems of BDI Agents," *Proceeding of the 7th European Workshop on Modeling Autonomous Agents in a Multi-Agent World,* 1996.

[LS08] J. Lian and S. Shatz, "A Modeling Methodology for Conflict Control in Multi-Agent Systems," *International Journal of Software Engineering and Knowledge Engineering*, Vol. 18, No. 3, May 2008, pp. 263-303.

[LSH07] J. Lian, S. Shatz and X. He, "Component Based Multi-Agent System Modeling and Analysis: A Case Study," *Proceedings of the International Conference on Software Engineering Research and Practice* (SERP), Vol. 1, pp. 183-189, Las Vegas, Jun. 2007.

[M89] T. Murata, "Petri Nets: Properties, Analysis and Application," *Proceeding of the IEEE,* Vol. 77, No. 4, pp. 541-580, Apr. 1989.

[MW97] D. Moldt and F. Wienberg, "Multi-Agent Systems Based on Coloured Petri Nets," *Proceeding of 18th International Conference on Application and Theory of Petri Nets,* 1997.

[NL05] E. Newcomer and G. Lomow, *Understanding SOA with Web Services*, Addison Wesley, 2005.

[S00] M. P. Singh, "Synthesizing Coordination Requirements for Heterogeneous Autonomous Agents," *Journal of Autonomous Agents and Multi-Agent Systems*, Vol. 3, No.2, pp. 107-132, 2000.

[S80] R. Smith, "The Contract Net Protocol: High-level Communication and Control in a Distributed Problem Solver," IEEE Transactions on Computers, Vol. 29, Issue 12, 1980.

[ST95] Y. Shoham and M. Tennenholtz, "On Social Laws for Artificial Agent Societies: Off-line Design," *Artificial Intelligence*, Vol. 73, pp. 231-252, 1995.

[XS03] H. Xu and S. Shatz, "A Framework for Model-Based Design of Agent-Oriented Software," *IEEE Transactions on Software Engineering*, Vol. 29, No. 1, pp. 15-30, Jan. 2003.

[WP04] M. Winikoff and L. Padgham, "The Prometheus Methodology," in *Methodologies and Software Engineering for Agent Systems. The Agent-Oriented Software Engineering handbook*, edited by F. Bergenti, M. Gleizes and F. Zambonelli, Kluwer Publishing, 1-4020-8057-3, Chapter 11, July 2004.

[ZJ01] F. Zambonelli, N. Jennings and M. Wooldridge, “Organizational Rules as an Abstraction for the Analysis and Design of Multi-Agent Systems,” *Journal of Knowledge and Software Engineering*, Vol.11, No.3, 2001.

[ZJW03] F. Zambonelli, N. Jennings and M. Wooldridge, “Developing Multiagent Systems: The Gaia Methodology,” *ACM Trans. on Software Engineering and Methodology*, Vol. 12, No.3, pp. 317-370, 2003.

Appendix

A: Correctness Comments for Level-1 Coordinator Generation Algorithm

Supposed that $s = \{b_i \mid b_i$ *is a binding policy*$\}$ denotes the set of binding policies included in a Level-1 coordinator.

For any binding transition *b = (Pre, Post)* not belonging to *s*, we denote its corresponding forward binding transition as *ft*, and it backward binding transition as *bt*. In the Level-0 coordinator, there exists one CPN place denoting *Pre*, and one CPN place denoting *Post*. The firing of *ft* is the only way for place *Pre* to obtain a token from *Post* and thereafter enforce *b.*

According to the Level-1 Coordinator Generation Algorithm in page 17, in the Level-1 coordinator, transitions ft and bt, and arcs *(Pre, ft)*, *(ft, Post)*, *(Post, bt)* and *(bt, Pre)* have been eliminated by the Level-1 coordination generation algorithm. Therefore, place *Pre* cannot obtain any token from *Post*, and the binding policy *(Pre, Post)* is therefore excluded from the Level-1 coordinator.

B: Correctness Comments for Priority Local Policy (Priority LP) Processing Algorithm

A priority binding policy *b1 > b2*, where *b1 = (pre1, post1)* and *b2 = (pre2, post2)*, is meaningful iff *post1* is identical to *post2*. Therefore, in a Level-1 coordinator, there will be CPN places to denote *pre1*, *pre2* and *post* respectively, for a valid priority binding policy *b1 > b2*.

We inherit other notations from the Priority Local Policy Processing Algorithm at page 19 in the rest of this appendix. In the Priority Local Policy Processing Algorithm at page 19, *v = 0* is the guard condition of *b2.ft*, and *b2.ft* is the only transition to enforce *b2*. Thus, if we can guarantee that *v = 0* iff there is no token from *pre1* currently in place *post* (indicating enforcing *b2* does not violate *b1*), we can ensure that *b1 > b2*.

In the Priority Local Policy Processing Algorithm, *v* is initialized as 0 and incremented whenever a token from *pre1* is deposited into *post*, and decremented whenever a token is removed from *post*. For this reason, *v = 0* iff there is no token from *pre1* and thereby this algorithm ensures that *b1 > b2*.

C: Correctness Comments for Proportional Local Policy (Proportional LP) Processing Algorithm

A proportional binding policy *b1|b2 = c*, where *b1 = (pre1, post1)* and *b2 = (pre2, post2)*, is meaningful iff *post1* is identical to *post2*. Therefore, in a Level-1 coordinator, there will be CPN places to denote *pre1*, *pre2* and *post* respectively for a valid proportional binding policy *b1|b2 = c*.

We inherit other notations from the Proportional Local Policy Processing Algorithm at page 21 for the rest of this appendix. In the Proportional Local Policy Processing Algorithm, *v >= c* is the guard condition of *b2.ft*, and *v < c* is the guard condition for *b1.ft*. Moreover, *b1.ft* is the only transition to enforce *b1,* and *b2.ft* is the only transition to enforce *b2*.

Also, the algorithm ensures that *v* is initialized as 0, incremented when a token from *pre1* is deposited into *post* by firing its forward transition and decremented when a token is removed from *post* and deposited into *pre1* through firing its backward transition. For those tokens from *pre2*, *v* is incremented *c* times when a token is deposited into *post* by firing its forward transition, and decremented *c* times when a token is removed via its backward transition. For this reason, *v = c* only when *b1* has been enforced *c* times. *v= c* will disable *b1.ft* and enable *b2.ft*. After *b2.ft* has been fired to enforce *b2*, *v* is reset to 0.

Therefore, the Proportional Local Policy Processing Algorithm ensures that *b2.ft* can be fired after *b1.ft* has been fired every *c* times. Since firing *b2.ft* enforces *b2* and firing *b1.ft* enforce *b1*, the algorithm ensures that $b1 \mid b2 = c$.

D: Correctness Comments for Universal Global Policy Processing Algorithm

A universal global policy $|p| < c$, mandates that the number of tokens never exceeds *c* in the CPN place representing *p*.

We inherit other notations from the Universal Global Policy Processing Algorithm at page 25 for the rest of this appendix. In the Universal Global Policy Processing Algorithm, $max < c$ is used as the guard condition associated with all the forward binding transitions relevant to *p*. So, when $max > = c$, all the forward binding transition will be disabled by the guard condition. Consequently, in this situation no more tokens can be deposited into place *p.*

The Universal Global Policy Processing Algorithm also ensures that when a token is deposited into place *p*, *max* is incremented, whereas *max* is decremented whenever a token is removed from place *p*. Therefore, the construction guarantees that $max = c$ when the place representing *p* has *c* tokens. Because all the forward binding transitions are disabled when $max = c$, no token will be deposited into *p* once *p* has reached the max number of tokens, unless the value of *max* is reduced due to token consumptions.

For the reasons above, the universal global policy processing algorithm can guarantee that the quantity of the specific resource tokens denoted by *p* will not exceed *c*, i.e., what a universal global policy $|p| < c$ has defined.