Component Based Multi-Agent System Modeling and Analysis: A Case Study[1]

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

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

Abstract - *We previously proposed a Colored Petri Net (CPN) based modeling methodology to model multi-agent systems. The methodology creates a component to describe the local behavior for each agent, and then concatenate all components to form a system model. In this paper, we focus on a case study to evaluate our modeling approach. The experiments in the case study are based on modeling and simulation of a multi-agent system for the Chicago downtown subway system.*

Keywords: Multi-Agent-System, Colored Petri Net, State Space Analysis, Traffic System Modeling.

1 Introduction

In a Multi-Agent System (MAS), multiple agents may work together to perform tasks or solve problems. Conflicts may occur in the runtime when multiple agents compete for external resources. To create MAS models for those dynamically coordinate agents to avoid run-time resource conflicts, we define a new feature called a potential arc that is used to extend traditional Colored Petri Net (CPN) models to support the agent behavior modeling in MASs [6]. Potential arcs serve to extend the traditional CPN model with an explicit support for distinguishing the representation of potential conflicts and real conflicts, and thereby avoid the need of eliminating all potential conflicts in the design stage. As a result, we can achieve the independence in local plan designs while still guarantee that the resulting MAS to be conflict-free. We select CPN as the modeling tool because CPN-based MAS models can support the simulation of dynamic execution of a MAS and the token driven mechanism in CPN can be used to describe the event triggering mechanism of conflicts. Also, the existence of many CPN based agent modeling techniques [4 and 5] indicates that CPN is an effective tool in MASs modeling.

Based on the potential arc concept, we previously also presented a potential arc based MAS modeling methodology [6]. The design of our modeling methodology embraces the principle of "separation of concerns" in the agent-oriented design, and focuses on modeling the possible behaviors of the MAS. As such, this work is complementary to other research efforts that model other aspects of a MAS [HM04]. The modeling methodology first creates the local plan for each agent independently, and then concatenates local plans together to form a global plan, which describes the whole MAS system. An important observation is that the global plan preserves all the elements from different local plans, and consequently preserves the agent features modeled by local plans. Therefore, the modification on a global plan can be easily mapped to local plan levels. Also, an agent's local plan can be modified or removed in a way that is transparent to other agents and be reconnected to an existing global plan as long as the potential arcs of the modified local plan remain the same as in the pre-modified local plan. The reconnection is done by repeating the coordinator and global plan generation algorithm on the revised local plans. This mechanism can lower the cost of model revision, as will be demonstrated by the case study.

In this paper, we focus on a case study to evaluate our modeling methodology. This case study is based on modeling and simulation-based analysis of Chicago Downtown Subway System. By using the potential arc notation, the system can be modeled in a modular way, where each agent is independently designed. Two experiments are included to reason about the model's behavior and evaluate our modeling technique. The first experiment showed that the resulting MAS model can be analyzed by a typical CPN analysis tool - CPN Tools [2] - to ensure a correct and efficient design. The second experiment demonstrated that our modeling technique can support model revision with low revision cost. Since the emphasis of this paper is not on the modeling technique but on the case study, we do not present details on such related work. Further details can be found in [6].

2 MAS modeling methodology

Our MAS modeling methodology first creates the local plan 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*

[1] This material is based upon work supported by the U. S. Army Research Office under grant number W911NF-05-1-0573.

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, in plan models 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.[2] To describe paths with external resources that may trigger a conflict, we introduce the concept of *potential arc*. 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.1 shows the local plan for agent AP. In Fig. 1, solid arcs represent regular CPN arcs, while dotted arcs denote the potential arcs. For a potential arc, the inscription above the arc is regular inscription and the inscription below the arc is 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 for the external resource to be released to the external environment.

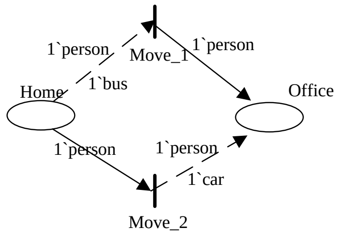


Fig. 1 PCPN-based local plan for a simple agent AP

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 a token "*person*" being 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 shows the example of how potential arcs are interpreted. In Fig. 2, the potential arc *(Home, Move_1)* from Fig. 1 is replaced by a regular arc, and place *R1*, outside the local plan model, is added to control the

[2] We assume familiarity with basic colored Petri net modeling.

enablement of *Move_1*. *R1* needs to produce the token "*bus*" so that transition *Move* can be enabled. The interpretation of potential arc (Move_2, Office) is similar. The details of the concatenation algorithm [6] are omitted due to the lack of space.

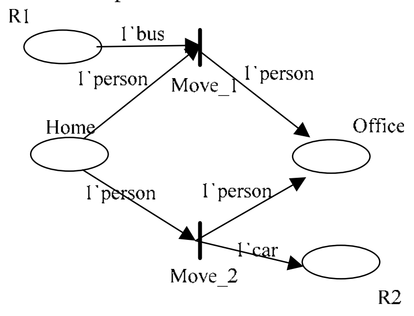


Fig. 2 The interpretation of Fig.1

Finally, the local plans and the coordinator will be concatenated to build a global plan - a CPN model for the whole MAS system. The details of the global plan generation algorithm [6] are omitted due to space concerns.

3 Case study: modeling

The city of Chicago currently operates seven subway trains in Chicago downtown area (See Fig. 3). These seven subway lines are denominated by seven different colors: *blueline*, *greenline*, *brownline*, *orangeline*, *purpleline*, *redline*, and *pinkline*. In the seven trains, blueline, redline and greenline operate bi-directionally, and the other four trains only operate in the uni-directional manner. Although different subway lines sometimes own different stations at the same area, these stations are all located within short physical distances. For instance, blueline and redline use two different subway stations at *Washington*, but these two stations stay within two minutes walking distance. Therefore, it is reasonable for us to neglect the commuting between subway stations at the same area, so we assume that all the subways share the same set of stations, identified by location names. We model twelve stations in the downtown area, denoted by location names: *Washington*, *Quincy*, *LaSalle*, *Library*, *Jackson*, *Monroe*, *Lake*, *State*, *Clark*, *Adams*, *Madison*, and *Randolph*. Commuters may travel from any one subway station to any of the other eleven stations. When traveling in the downtown area, a commuter generally takes subways, but he/she may also walk when the distance between two stations is short.

Based on the scenario above, we can develop a model for the Chicago subway system. The model illustrates the influence of subway operations on downtown commuting. From the software life-cycle point of view, the development of a MAS should cover the following phases: requirement capture, design, implementation and deployment. During the requirement capture phase, the scenario to model is analyzed so that the stakeholders can be identified. Furthermore, the behavior of each stakeholder is defined and each stakeholder is assigned an agent role in the MAS, based on its behavior pattern. Although our modeling technique focuses on the design phase, we need to inherit the modeling requirements from the requirement capture phase, which include the role, the quantity and the informal description of each agent. Also, a universal vocabulary should be identified to serve as the communication language for inter-agent resource sharing. Clarifying the vocabulary can avoid many problems typically caused by the assumption that different agents in the MAS can understand each other automatically [3]. Requirements capture, implementation and deployment are beyond the scope of our modeling technique and have been covered by other state-of-the art research, such as Tropos [1].

Section 3.1 introduces the modeling requirements and the vocabulary, Section 3.2, 3.3 depicts the agent models, and Section 3.4 provides a partial coordinator. The global plan is a concatenation of local plans and the coordinator based on the interpretation mechanism presented at Section 2. For this reason, the global plan does not provide extra info that cannot be reasoned from the local plans and the coordinator. Therefore, the global plan is omitted to alleviate the space concern.

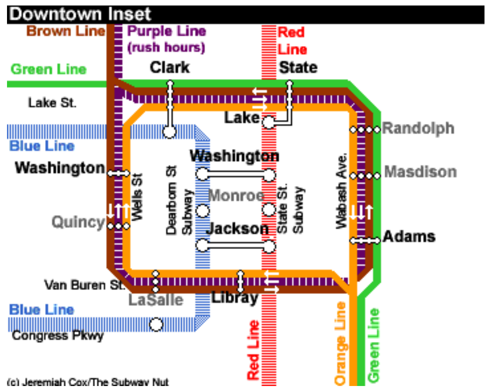
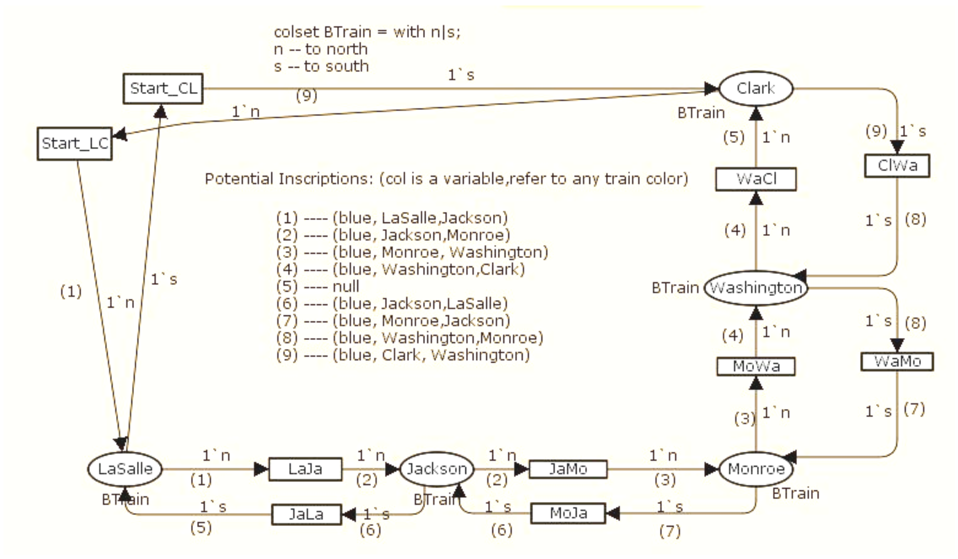


Fig. 3 Chicago Downtown Subway Map

3.1 Modeling requirements and the vocabulary

We design eight agent types in our system: seven *train dispatchers* and a *commuter monitor*. Quantity of each agent is one. Each train dispatcher is responsible for dispatching a specific train along the predefined route. The commuter monitor provides a routing map for commuters to travel between the twelve downtown stations. The commuting is mainly driven by trains. Besides, we also allow commuters to walk on foot



colset BTrain = with n|s;
n -- to north
s -- to south
Start_CL
Start_LC
Clark
BTrain
WaCl
ClWa
Potential Inscriptions: (col is a variable,refer to any train color)
(1) ---- (blue, LaSalle,Jackson)
(2) ---- (blue, Jackson,Monroe)
(3) ---- (blue, Monroe, Washington)
(4) ---- (blue, Washington,Clark)
(5) ---- null
(6) ---- (blue, Jackson,LaSalle)
(7) ---- (blue, Monroe,Jackson)
(8) ---- (blue, Washington,Monroe)
(9) ---- (blue, Clark, Washington)
Washington
MoWa
WaMo
LaSalle
LaJa
Jackson
JaMo
Monroe
JaLa
MoJa
1`n
1`s

Fig. 4 The Local plan for blueline train dispatcher

between two pairs of closely located stations, *(Clark, Lake)* and *(State, Lake)*.

The subway train is modeled as the shared resource, dispatched by train dispatchers. When a train is moving, the source and destination station changes dynamically. These two stations are used to identify the moving direction of a train and the current stop of the train, and they further influence the movement of commuters. Consequently, we incorporate the source station and destination station into the vocabulary definition. The vocabulary is denoted as a color set *RESOURCE*, and defined as follows by CPN Tools:

*colset RESOURCE = product Color * Loc * Loc*;

For example, *(blue, Washington, LaSalle)* denotes the blueline train at *Washington* station, and its next station is *LaSalle*. The commuters in *Washington* station can thereby take this train to *LaSalle* station.

Although these local plans all need to use a common vocabulary in order to share resources, we should remember that the local plan of each agent is independently developed. Therefore, the commuter monitor is unaware of the subway routes, and the design of train dispatcher is independent of the commuter routing map.

3.2 Train dispatcher models

The local plan of each specific train dispatcher agent needs to incorporate the train operation routes; therefore, they are developed based on the corresponding subway routing map. The behavior of a train is composed of a series of directed inter-station moving actions. Due to the fact that Chicago subway trains operate in an acyclic manner, there is only one directed path between every two stations. Consequently, only one path exists for each action in the local plan. For example, Fig. 4 shows the local plan for blue line train dispatcher. Moving from *LaSalle* station to *Jackson* station is an action of the blueline dispatcher, and there is only one path for this action, since there is only one acyclic railway lane available for blueline between LaSalle and Jackson according to the subway routing map. This path is modeled by transition *LaJa* and its surrounding arcs.

In Fig. 4, the arcs with two inscriptions are potential arcs, where potential inscriptions are embraced by parenthesis. Token *n* and *s* are used to distinguish the north-bound and south-bound trains. The movement of the train is modeled by firing a corresponding transition. As we know, the source and destination station keep changing when the train is moving. Henceforth, the train dispatcher agent model keeps updating the source and destination station by designating the source and destination station at the incoming and out-going potential inscriptions. For instance, potential inscriptions *(1)* and *(2)* indicate that firing transition *LaJa* consumes the resource *(blue, LaSalle, Jackson)* and produce the resource *(blue, Jackson, Monroe),* so that the source station and destination station is updated.

It is worth noting that transitions Start_CL and Start_LC are added to Fig. 4 to connect the two terminal stations *LaSalle* and *Clark* to model the following dispatching mechanism: when a north-bound (south-bound) train runs to its end station, a new train is dispatched from the starting station. The local plans for other subway trains are omitted due to the lack of space.

3.3 The commuter monitor model

The commuter monitor models the following commuting behavior: *a number of commuters travel by train or on foot between the twelve Chicago downtown subway stations*. The commuter monitor designer acknowledges the fact that downtown locations are intensively connected by subways station but has no knowledge about subway routing maps. The local plan for the commuter monitor is designed to include the description of subway stations and paths to take various actions for commuters to move among stations. Each subway station is modeled by a CPN place. An action is defined as the movement of a number of commuters from one subway station to another. Also, each token in the commuter monitor represents an arbitrary commuter, due to the fact that all commuters share the same commuting behavior pattern. For example, moving passengers from station *Lake* to *State* is an action. The local plan models one or more paths for every action. A path can be a direct path between two stations modeled by a corresponding transition (referred as *path transition*) and its surrounding arcs in the local plan, or an indirect path via one or more third-party stations. Indirect paths can be obtained by studying the topology of the local plan. Fig. 5 shows the local plan of the commuter monitor. In Fig. 5, there are twelve CPN places, modeling the twelve subway stations. Path transitions are used to represent the commuter movement from one station to another, with the arcs denoting the moving direction. Two types of path transitions are included in the local plan: a *train transition* represents traveling by train and is connected by potential arcs to specify the related external resource - train. A train transition will be enabled under two conditions: there are tokens (representing commuters) in its source place and there exists the related train resource (specified by the potential inscriptions). A *walking transition* represents traveling on foot and is connected by regular transitions, since no external resources will be involved.

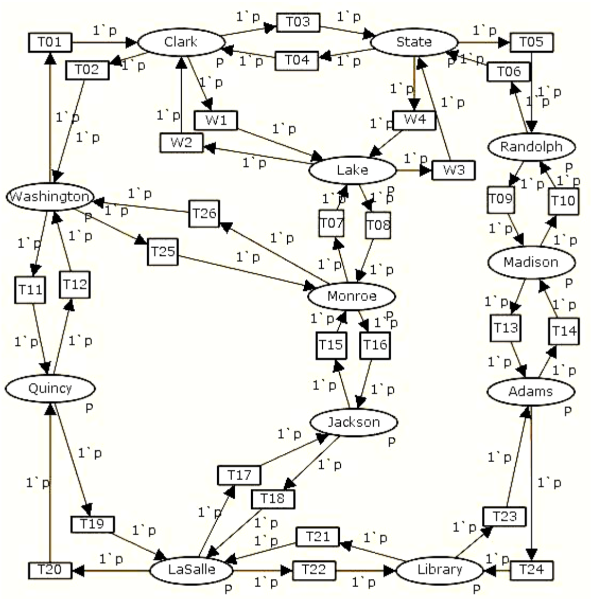


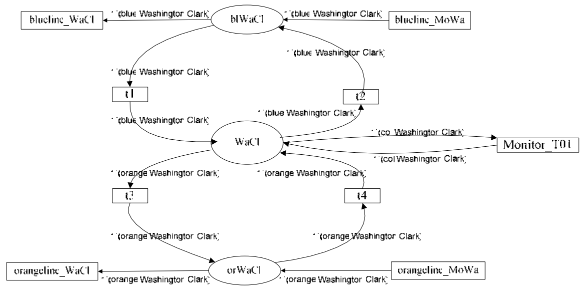
Fig. 5 Local plan for the Commuting Monitor

All the transitions in Fig. 5 are train transitions, except for the following four walking transitions: *W1*, *W2*, *W3* and *W4*. For other train transitions between two stations, the incoming and out-going potential arcs carry the same potential inscriptions -- denoting the train from the corresponding source station to the destination station. Each potential inscription is specified as *(col, t's source station, t's destination station)*. Here *t* is the connected transition for the corresponding potential arc, and *col* is a variable representing the train color. The binding of *col* is unresolved in the local plan, which means that the passenger is allowed to take any available train from the source station to the destination station. For example, the incoming and out-going potential arcs connected to transition *T01* carry the same potential inscription - 1`(*col, Washington, Clark*). Potential inscriptions are omitted from Fig. 5 due to the large quantity of potential arcs. Both direct and indirect paths are modeled by Fig. 5. Take the *Lake* - *State* movement action as an example. Walking through *W3* is a direct path. An alternative indirect path can be walking from *Lake* to *Clark* via W2, then take train via T03 from *Clark* to *State*.

Note that the local plan is designed according to the downtown street map, not the subway routing map. Therefore, not all subway routes are used by the commuters in the local plan, and not all paths in the local plan are connected by subway lanes. Also note that our modeling technique focuses on presenting a modeling technique, so the topic of studying the local plan topology to calculate indirect paths is out of the scope of this paper.

3.4 The coordinator and the global plan

The coordinator provides a perspective to inspect the resource utilization. Due to the large size of the coordinator, we only show part of it. Fig. 6 is a partial coordinator illustrating the subway operations from *Washington* to *Clark*. In Fig. 6, the agent name is added as a prefix to each transition inherited from local plans to distinguish transitions from different agents. For example, transition *blueline_WaCl* is inherited from transition *WaCl* in the local plan of the blueline dispatcher agent (Fig. 4). According to our scenario, trains at *Washington* heading for *Clark* can carry passengers. From Fig. 6, we can see that *Monitor_T01* (inherited from T01 in Fig. 5) is enabled when there is a token (model a *Washington* to *Clark* train) in place *WaCl*. Compared with Fig. 5, we can see that firing *Monitor_T01* (*T01* in Fig. 5) can move the tokens (model passengers) from place *Washington* to place *Clark*. Both orange line and blue line operate from *Washington* to *Clark*. Therefore, we have both *blWaCl* (modeling the resource *(blueline, Washington, Clark)*) and *orWaCl* (modeling the resource *(orangeline, Washington, Clark)*) in the coordinator. The four binding transitions *t1* to *t4* make place *WaCl* capable of obtaining tokens *(blue, Washington, Clark)* and *(orange, Washington, Clark)* so that both orange line and blue line are available for the passengers from *Washington* to *Clark*. Whether a specific train is available falls under the control of the transitions inherited from the local plan of a train dispatcher. For example, transition *blueline_Mowa* and *blueline_WaCl* are inherited from the blueline dispatcher to control the availability of blueline trains for passengers willing to travel from Washington to Clark. Firing *blueline_MoWa* (blueline moves from Monroe to Washington) deposits a token *(blue, Washington Clark)* to place blWaCl, so that place *WaCl* can obtain this token by transition *t1* to enable transition *Monitor_T01*. If the token *(blue, Washington, Clark)* is returned to *blWaCl* by *t2*, transition *blueline_WaCl* can be fired (blueline leaves *Washington* and arrives at *Clark*) so that blueline is no longer available for passengers at *Washington* station.


blueline_WaCl
1`(blue Washington Clark)
blWaCl
1`(blue Washington Clark)
blueline_MoWa
1`(blue Washington Clark)
1`(blue Washington Clark)
t1
t2
1`(blue Washington Clark)
1`(blue Washington Clark)
1`(co Washington Clark)
WaCl
Monitor_T01
1`(col Washington Clark)
1`(orange Washington Clark)
1`(orange Washington Clark)
t3
t4
1`(orange Washington Clark)
1`(orange Washington Clark)
orangeline_WaCl
orWaCl
orangeline_MoWa
1`(orange Washington Clark)
1`(orange Washington Clark)
Fig. 6 The Partial Coordinator

4 Case study: experiments

One of the advantages of model development is to help designers to study the properties of the target system to ensure a correct and efficient design, which should meet some key requirements, such as liveness, deadlock freeness, and concurrency. Moreover, no real systems are ever static. Many external factors may change the behavior of the system. For a subway system, accidents, construction or maintenance may force one or more trains to stop operations temporarily or permanently. As a result, one or more station may need to be closed. It is desirable that the model can also be revised with relatively low cost to adopt the system modification, so that we can study the modified system. Through our experiments, we can demonstrate that our modeling technique can help designers to study system properties and provide flexibility in model revision to lower revision cost.

All experiments are performed by importing the resulting global plan into CPN Tools 2.0 [2] under Microsoft Windows XP. The global plan is simulated to study the run-time behavior of the MAS, which is to be developed based on the global plan. During the simulation, if multiple transitions are enabled at a specific step, CPN Tools randomly select an enabled transition to fire. We are aware of the fact that simulation-based analysis cannot provide results as strong as formal verification (over a full state-space). Still, simulation is widely recognized as a valuable analysis method, which can help identify design errors without expensive state-space analysis.

CPN Tools provide a set of mechanisms to facilitate the analysis of Petri net models, and these mechanisms can tailor the simulation traces for various analysis purposes. In CPN Tools, a *monitor* is a mechanism that is used to observe, inspect, control, or modify a simulation of a colored Petri net. Monitors can inspect both the markings of places and the occurring binding elements during a simulation, and they can take appropriate actions based on the observations.

Experiment 1: When designing the commuter monitor, we acknowledge that the twelve stations are highly connected by subways. Therefore, the commuter monitor models many direct paths by train transitions (transitions that model trains). However, not every two subway stations are directly connected by trains. Consequently, some train transitions in the commuter monitor may be redundant since it represents a path that will never be taken. These redundant path transitions may lead to low efficiency of commuting (Because it increases the possibility of forcing commuters to switch to an alternative path). Although the existence of redundant path transitions does not violate our promise to guarantee design independence, it is desirable to identify and eliminate these redundant transitions for efficiency concerns. In this experiment, we analyze the global plan by simulation to identify these redundant train transitions. We use a type of monitor - *count transition occurrences monitor* - to calculate the number of times a particular transition occurs during a simulation. We install a count transition occurrence monitor for each train transition inherited from the commuter monitor. For the experiment, we initialized the global plan by assigning 100 tokens to each place that represents a subway station. Each simulation run uses 100,000 steps (transition firings) and we repeat the experiment five times. Fig. 7 illustrates the occurrence times for each train transition. In Fig. 7, *exp1*, *exp2*, *exp3*, *exp4* and *exp5* represent the five simulation instances of the experiment respectively. The transition occurrence times for each transition are shown as bars, and the experiment results for the five experiments are stacked together to save horizontal space (We need to show data for 26 train transitions). If a train transition never occurs during extensive random simulation, we can say that the train transition is redundant because no trains are available at the corresponding path. Fig. 7 clearly shows that transitions *T07*, *T08* and *T24* are never fired in these five experiments. Another observation is that the results from the five experiment instances are rather close, which indicates that 100,000 steps is sufficient for our experiment. Therefore, we can have high confidence in our result that *T07*, *T08* and *T24* are redundant. The redundancy of *T07*, *T08* and *T24* can be justified by the following facts: *T07* and *T08* represent the direct paths between *Lake* and *Monroe* for the commuter model. However, these transitions can never be fired since there is no train that directly operates between these two stations in the train dispatcher models we are using. *T24* models the path from *Adams* to *Library*, but we did not model any train operation from *Adams* to *Library* for the convenience of Experiment 2. After eliminating *T07*, *T08* and *T24* from the local plan of the commuter monitor, we generate the new global plan. In this new global plan, no redundant train transitions exist.

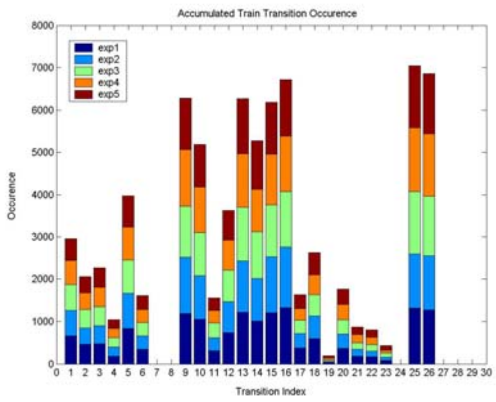


Fig. 7 Accumulated Train Transition Occurrences

Experiment 2: In this experiment, we demonstrate that our modeling technique can support model revision with low revision cost. We examine whether each station is still reachable from any other eleven ones if one of those trains stops running. As our example of this type of "sensitivity analysis," let us consider the removal of brownline agent and generate a new global plan using the other six train dispatchers and the revised commuter monitor from Experiment 1. In the global plan, we place 100 tokens at the place representing *Washington* station and run simulations for the global plan. Still, each simulation contains 100000 steps and the experiment is repeated five times. We use *marking size monitors* to extract the maximum number of tokens at each station place during a simulation. The results can be found in Fig. 8.

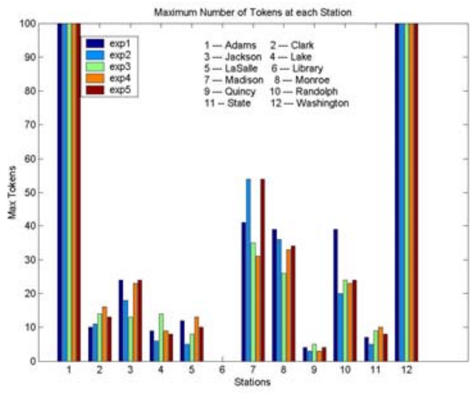


Fig. 8 Maximum Number of Tokens at each Station

Similar to Fig. 7, *exp1*, *exp2*, *exp3*, *exp4* and *exp5* in Fig. 8 represent the results from five experiment instances, respectively. For each place that models a subway station, the maximum number of tokens held by the place is shown in Fig. 8. (Results from the five experiment instances are grouped not stacked, since we only need to show 12 values on the x-axis, instead of 26, as in Fig. 7; so we do not need to save horizontal space by stacking) Fig. 8 indicates that commuters are unable to reach *Library* from *Washington*, since there never exists any token during the simulation process. This observation is attributed to the fact that the suspension of brownline subway train and the lack of train operation from *Adams* to *Library* make all the paths from *Washington* to *Library* unreachable.

The model revision is done by simply removing the brownline dispatcher and then running the automatic coordinator generation and global plan generation procedure with the remaining seven agents' local plans (six train dispatchers and the commuter monitor). Since none of the seven remaining agents needs to be redesigned and we do not need to change any model generation algorithm, it is clear that we incur little revision cost beyond that of removing one local plan.

5 Conclusion

In this paper, we focused on presenting a case study to illustrate a modeling methodology for multi-agent systems. The case study models the Chicago downtown subway system and includes two experiments to evaluate our overall approach. Future research will focus on defining more flexible coordinators and tool development.

6 References

[1] P. Bresciani, P. Giorgini, F. Giunchiglia, J.Mylopoulos and A. Perini. "TROPOS: An Agent-Oriented Software Development Methodology," *Journal of Autonomous Agents and Multi-Agent Systems*, 8(3):203-236, 2004.

[2] CPN Tools (Version 2.0) help file, 2006.

[3] L. Cysneiros and E. Yu. "Requirements Engineering for Large-Scale Multi-agent Systems," *Lecture Notes in Computer Science*, Vol. 2603, pp. 39-56, Springer, 2003.

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

[5] 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, 2004.

[6] J. Lian and S. Shatz. "A Modeling Methodology for Conflict Control in Multi-Agent Systems," To appear in *International Journal of Software Engineering and Knowledge Engineering*, 2007.