An Agent Mediated Approach To Dynamic Change in Coordination Policies

Prasanta Bose[1] and Mark G. Matthews[2]

[1] Information and Software Engineering Department, George Mason University, Fairfax, VA 22030
bose@mgfairfax.rr.com
[2] The MITRE Corporation, 1820 Dolley Madison Blvd., McLean, VA 22012
mmatthew@mitre.org

Abstract. Distributed information systems for decision-support, logistics, and e-commerce involve coordination of autonomous information resources and clients according to specific domain independent and domain dependent policies. A major challenge is handling dynamic changes in the priorities, preferences, and constraints of the clients and/or the resources. Addressing such a challenge requires solutions to two problems: a) Reasoning about the need for dynamic changes to coordination policies in response to changes in priorities, preferences, and constraints. b) Coordinating the run-time assembly of policy changes in a dependable manner. This paper introduces the NAVCo approach to address these problems. The approach involves exploiting negotiation-based coordination to address the first problem and model-based change coordination to address the second problem. These two key features of the approach are well suited for realization using an agent-based architecture. The paper describes the architecture with specific emphasis on the analysis and design of the agent specifications for negotiation and change coordination.

1 Introduction

Distributed information systems for decision-support applications, logistics, and e-commerce involve coordination of autonomous information resources and clients according to specific domain independent and domain dependent policies. A major challenge is handling dynamic changes in the preferences and constraints of the clients and/or the resources.

Consider the domain of distributed decision-support applications where autonomous information resources are coordinated to meet the information demands of client specific decision-support views. In such a domain, there is continuous change in Quality of Service (QoS) properties and constraints of the clients, information resources, and shared communications infrastructure. Current architectures for coordination of distributed information resources to support client specific views are static in nature; that is the coordination policies cannot be dynamically changed to meet changing demands. As an example, consider the following scenario from the supply-chain application domain:

A decision-support view for inventory management is maintained from multiple autonomous information resources within a supply-chain. Customer order information from customer sites, product assembly information from manufacturer sites, and parts inventories from parts supplier sites are configured to support an order-fulfillment view used by inventory managers of the suppliers and consumers. As orders, product assembly requirements, and parts inventories constantly change, changes in the view must be coordinated to achieve consistency and to support management decisions.

There are multiple view maintenance policies available to support the above inventory management task. Typical systems are based on static architectural decisions (on view maintenance policies) at design time by considering tradeoffs between consistency, communications costs, and processing costs. Suppose a high-cost complete consistency view maintenance policy was selected for implementation at design time. Further suppose that several inventory managers are simultaneously executing intensive on-line analytical processing (OLAP) queries against the order fulfillment view. The queries are competing with the view maintenance policy for system resources. Under these conditions, both the queries and the view maintenance task are likely to suffer from poor performance. Short of shutting down, reconfiguring, and restarting the system, current architectures have no way of prioritizing preferences and dynamically responding to changing preferences and constraints.

1.1 Self-Adaptive Software: Requirements

A key observation to be made from the above discussion is that automated and dynamic approaches to addressing the problem of changing preferences and constraints require architectures and adaptive mechanisms that achieve dynamic self-design in response to changing preferences and constraints. The four major capabilities of such self-adaptive software systems are: i) *Detecting a change in context or a change in needs.* The system should be able to monitor its behavior and detect deviations from its commitments or the presence of new opportunities. It should be able to accept new needs from external sources and evaluate for deviations with respect to current commitments. ii) *Knowing the space of adaptations*. It must have knowledge of the space of self-changes it can choose from to reduce deviations. iii) *Reasoning for adaptation decision.* It should be able to reason and make commitments on the self-changes and commitments on revised goals. iv) *Integrating the change*. It should be able to package the change if required and perform assembly/configuration coordination to insert the change into the existing system in a dependable manner with minimal disruption to existing behaviors.

1.2 NAVCo Approach: Key Ideas

The NAVCo approach described in this paper considers a family of adaptive systems for information view management. The key distinctive features of the approach are: a) Changes in committed preferences and context assumptions trigger the adaptation process. b) An adaptation space defined by a set of view maintenance policy objects that forms the basis of design for a set of middleware coordination agents. c) Reasoning for change accomplished through a negotiation based process involving the clients and information resource agents. d) Use of verified assembly plans for change integration. The above features of the NAVCo approach are well suited for realization using agent-based concepts and an agent-based architecture.

The following sections of the paper describe the agent-based architecture. The paper focuses primarily on the negotiation-based coordination used to reach a change decision and the consequent coordination used to incorporate the change decision. In the context of NAVCo, a change decision equates to a decision to switch between view maintenance policies at run time.

2 Multi-Resource View Maintenance Policies: Background

Multi-resource view maintenance falls within the domain of distributed decision-support database systems. A simplified model of this domain is illustrated in Figure 1. As illustrated in Figure 1, a view (V) is maintained from a set of autonomous data sources (S_1, S_2,…,S_n). The view is a join of relations (r_1, r_2, r_n) within the data sources. The update/query processor and view maintenance policy components execute a distributed algorithm for incrementally maintaining the view. As data within a source changes, the associated update/query processor sends notification of the update to the view maintenance policy (VMP) component in Figure 1, which in turn queries the other sources to compute the incremental effect of the source update. After the incremental effect of the update has been computed, it is propagated to the client view. Client applications, such as on-line analytical processing (OLAP) and data mining, execute queries against the view. The data sources also support transactional environments, which result in updates to source relations that participate in the view.

The agent-based architecture presented in this paper focuses on providing mechanisms to allow run-time switching of view maintenance policies. Four VMPs are briefly discussed and compared in this section. A complete description of these policies can be found in [1, 21].

The Strobe algorithm is an incremental VMP that achieves strong consistency. The Strobe algorithm processes updates as they arrive, sending queries to the sources when necessary. However, the updates are not performed immediately on the materialized view (MV); instead, a list of actions (AL) to be performed on the view is generated. The MV is updated only when it is certain that applying all of the actions in AL (as a single transaction at the client) will bring the view to a consistent state. This occurs when there are no outstanding queries and all received updates have been processed.

The Complete-Strobe (C-Strobe) algorithm achieves complete consistency by updating the materialized view after each source update. The C-Strobe algorithm issues compensating queries for each update that arrives at the VMP between the time that a query is sent from the VMP and its corresponding answer is received from a source. The number of compensating queries can be quite large if there are continuous source updates.

The SWEEP algorithm achieves complete consistency of the view by ordering updates as they arrive at the VMP and ensuring that the state of the view at the client preserves the delivery of updates. The key concept behind SWEEP is on-line error correction in which compensation for concurrent updates is performed locally by using the information that is already available at the VMP. The SWEEP algorithm contains two loops that perform an iterative computation (or sweep) of the change in the view due to an update.

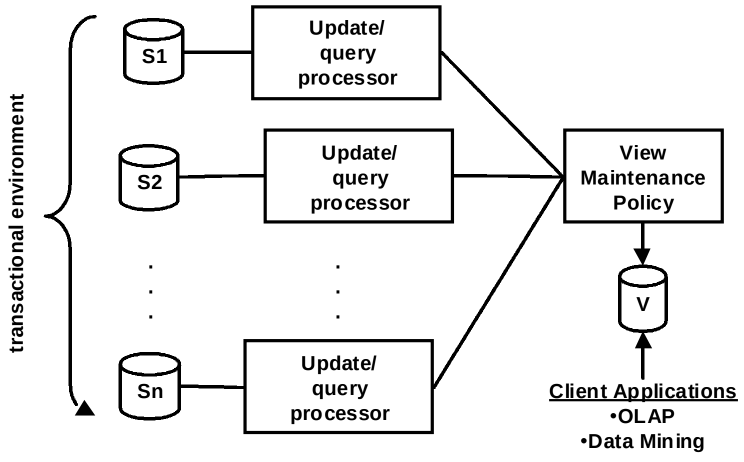


Fig. 1. Distributed Decision-Support Database System Domain

The Nested SWEEP algorithm is an extension of the SWEEP algorithm that allows the view maintenance for multiple updates to be carried out in a cumulative fashion. Nested SWEEP achieves strong consistency by recursively incorporating all concurrent updates encountered during the evaluation of an update. In this fashion, a composite view change is computed for multiple updates that occur concurrently.

2.1 Cost-Benefit Analysis

The performance of VMPs can be compared based on the communications and processing costs required to maintain a certain level of consistency. Communications costs can be measured with respect to the number and size of messages required per update. Processing costs can be measured with respect to the processing burden that the algorithm places on both the client and the data sources.

Table 1 compares the communications and query processing cost of four VMPs. The cost of the algorithms is dependent on the number of data sources, *n*. The costs of the C-Strobe and Nested SWEEP algorithms are highly dependent on a workload characterization factor, *a* where $0 \leq a \leq 1$, which reflects the rate of updates received. If updates arrive infrequently $a=0$ and if updates arrive continuously $a=1$. The client processing cost of a delete update in the Strobe and C-Strobe algorithms is highly dependent on the number of pending updates, *p*. The costs in Tables 1 and 2 depict the case in which the VMP components are co-located with the client.

Table 1. VMP Cost Comparison

Algorithm	**Update Type**	**Comm. Cost**	**Client Cost**	**Server Cost**
Strobe	Delete	1	1+p	0
	Insert	2n-1	(n-1)+1	1
C-Strobe	Delete	1	1+p	0
	Insert	2(n-1)+2a(n-1)!+1	(n-1)+a(n-1)!+1	1+a(n-2)!
SWEEP	delete/insert	2n-1	(n-1)+1	1
Nested SWEEP	delete/insert	2(1-a)(n-1)+1	(1-a)(n-1)+1	(1-a)

As illustrated in Table 1, the cost of the algorithms is highly sensitive to the volume and types of updates. To illustrate, consider the inventory management scenario introduced in Section 1. Further assume that there are four information resources, one client view, and the following dynamic workload:

- Period 1 -high volume, high insert (100 inserts, 0 deletes, X seconds)
- Period 2 - low volume, balanced (50 inserts, 50 deletes, 3X seconds)
- Period 3 - medium volume, high delete (0 inserts, 100 deletes, 2X seconds)

The cost of each algorithm over these periods can be calculated using the formulas in Table 1. The value of the parameter *p* is assumed to be 0 for low traffic, 10 for medium traffic, and 100 for high traffic. The value of the parameter, *a,* is assumed to be 0 for low traffic, 1/3 for medium traffic, and 1 for high traffic. The cost of the four algorithms over Periods 1-3 is illustrated in Table 2.

Table 2. Cost in Inventory Management Scenario

Algorithm	**Comm Cost**	**Client Cost**	**Server Cost**
Strobe	1200	1750	150
C-Strobe	2400	2350	350
SWEEP	2100	1200	300
Nested SWEEP	1250	775	158
Example 1	750	1525	75
Example 2	950	625	108

Currently a single algorithm is selected at design time and cannot be changed without shutting down and reconfiguring the system. Design-time tradeoffs must be made with respect to consistency versus client, server, and communications costs. The design-time decision can have a profound effect on the processing and communications requirements to support the view. If, however, the algorithm can be dynamically changed at run-time, these tradeoffs can be made continuously as preferences and constraints change. As illustrated in the two examples at the bottom of Table 2, the ability to dynamically switch algorithms can result in significant cost savings and improved performance in a constrained environment.

Example 1 shows that communications cost can be minimized by initially implementing the Nested SWEEP algorithm and then dynamically switching to the Strobe algorithm between periods 1 and 2. This results in a cost reduction of 450 messages over a static implementation of the Strobe algorithm. This frees up valuable shared communications resources for more critical applications.

Example 2 shows that client processing cost can be minimized by implementing the Nested SWEEP algorithm during periods 1 and 3, and the Strobe algorithm during period 2. This results in a cost reduction of over 1000 queries over a static implementation of the Strobe algorithm. This frees up valuable resources for processing-intensive analysis queries and results in a significant performance improvement for analysis users.

3 NAVCo Agent-Based Architecture

The NAVCo approach to adapting view maintenance policies in response to changes in the needs of the clients or changes in the constraints imposed by the resources is based on negotiation reasoning between the clients and resources followed by dynamic change coordination. The approach is based on a layered architecture with agent-based middleware in each layer. The layers in the architecture, shown in Figure 2 as a UML class diagram, separate the concern for policy change reasoning, policy change coordination and application specific information view maintenance based on a policy. The architectural components and connections are modeled as stereotypes of the UML class. For the sake of brevity, we limit our discussions in the rest of the paper to an example involving switching between the Strobe and C-Strobe policies described in the previous section. The architectural components in each layer, the connections between the components and the connections between the layers are briefly described below.

3.1 Negotiation Coordination Layer

The key component types of the negotiation coordination layer (bottom layer in Figure 2) are a negotiation facilitator agent (NFA), a client negotiation agent (CNA), and resource negotiation agents (RNAs) that communicate via a shared data space called the Negotiation space (NSpace). The CNA and RNAs provide an agent-oriented negotiation interface to the client and resources. The negotiation is based on a formalization of the WinWin model [2, 3]. The communication of a negotiated change decision from the negotiation layer to the change coordination layer is via the shared Nspace.

3.2 Change Coordination Layer

The change coordination layer (middle layer in Figure 2) performs the actions required to dynamically switch between view maintenance policies in response to a communicated switching decision from the negotiation coordination layer. The change coordination layer consists of a Strobe tracking agent (STA), a C-Strobe tracking agent (CTA), a change coordination agent (CCA), and a shared data space (CSpace). There is one tracking agent per scheduler agent in the application coordination layer. A tracking agent interacts with the monitoring interface of its associated scheduler agent via the change coordination event channel (CChan). Each time a scheduling agent experiences a qualitative change in one of its state variables, the change is propagated to the associated tracking agent which abstractly tracks the execution state of the scheduling agent. The CCA accepts the communicated switching decision from the NFA via the NSpace and, based on the current tracking information maintained by the tracking agents, executes the set of configuration and control actions required to bring about the dynamic switching between VMP scheduling agents within the application coordination layer. The CCA interacts with the control interface of the scheduling agents via the CChan. The CSpace is used to communicate the tracking knowledge from the tracking agents to the CCA.

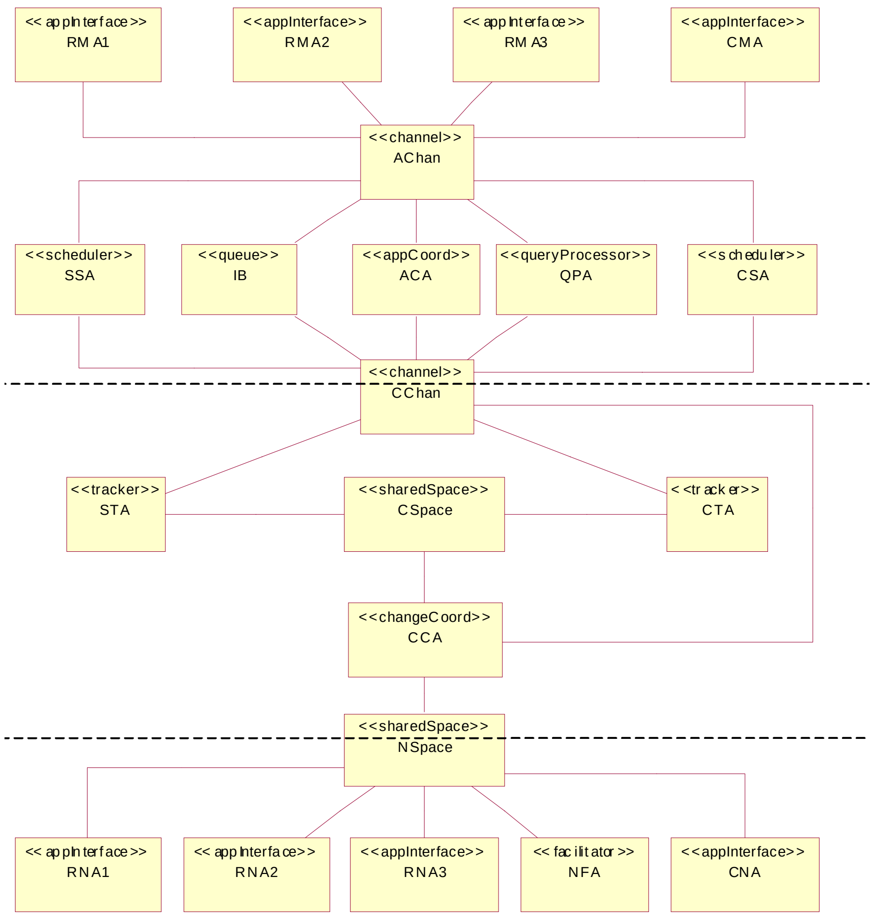


Fig. 2. The NAVCo Agent-based Architecture

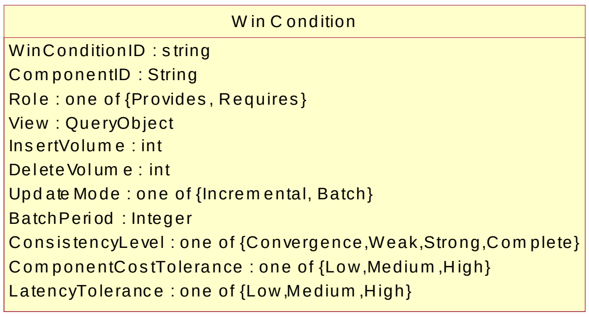
3.3 Application Coordination Layer

The application coordination layer (top layer in Figure 2) consists of agent representatives of the application specific service components that act to perform view maintenance. The application component interfaces necessary to support distributed view maintenance are achieved by the resource and client manager agents (RMA and CMA respectively) as depicted in Figure 2. The RMAs provide the functionality of the update/query processor depicted in Figure 1. To allow flexible switching of view maintenance policies, the activities of a view maintenance policy are decomposed and realized by a set of modular agents (scheduler, queue, and query processor) that allow reuse and flexible switching by localizing the change element to the scheduler agent. The scheduler agents (SSA for Strobe and CSA for C-Strobe) dynamically schedule updates and queries according to a specific VMP. The query processor agent (QPA) executes a query processing schedule. The input buffer (IB) component queues update tasks to be scheduled by the scheduling agent. An application coordinator agent (ACA) delegates view maintenenace tasks (in terms of updates) to the active scheduler agent. The agents interact via the application coordination event-channel (AChan). The SSA, CSA, IB, and QPA each contain a control interface and associated methods to allow the change coordination and control agent to perform configuration and control actions. The SSA and CSA also include a monitoring interface and associated methods to allow tracking agents to track policy execution states.

4 Design and Analysis of Negotiation Coordination Layer

The model for negotiation coordination used in our approach is based on the WinWin [2, 3] model used in multi-agent (representing stakeholders) requirements negotiation. In such a model, the participating agents collaboratively and asynchronously explore the WinWin decision space that is represented by four main conceptual artifacts: i) WinCondition - capturing the preferences and constraints of a participant. ii) Issue - capturing the conflict between WinConditions or their associated risks and uncertainties. iii) Option - capturing a decision choice for resolving an issue. iv) Agreement - capturing the agreed upon set of conditions which satisfy stakeholder WinConditions and/or capturing the agreed options for resolving issues. The artifacts specify the message objects passed between the agents. The object model for the WinCondition object developed for negotiating VMPs is shown in Figure 3. The object explicates attributes relevant to expressing preferences and constraints for the distributed view maintenance problem.

NAVCo incorporates three types of negotiation reasoning schemes that extend the WinWin model to consider a reactive model of negotiation. The first method, used during the initial establishment of the task and for negotiation of the initial policy, takes a task-driven approach and is triggered when a new client WinCondition is submitted. As illustrated in Table 3, the client initiates the task through submission of a WinCondition containing the task parameters and any preferences and constraints. The second method, depicted in Table 4 and used for run-time dynamic renegotiation of policies, is conflict driven and is triggered by changes in preferences and constraints. In this scheme any team participant may submit a revised WinCondition based on changing component preferences and constraints.



WinCondition

WinConditionID : string
ComponentID : String
Role : one of {Provides, Requires}
View : QueryObject
InsertVolume : int
DeleteVolume : int
UpdateMode : one of {Incremental, Batch}
BatchPeriod : Integer
ConsistencyLevel : one of {Convergence,Weak,Strong,Complete}
ComponentCostTolerance : one of {Low,Medium,High}
LatencyTolerance : one of {Low,Medium,High}

Fig. 3. The WinCondition Object Model

Table 3. Task Driven Negotiation Protocol

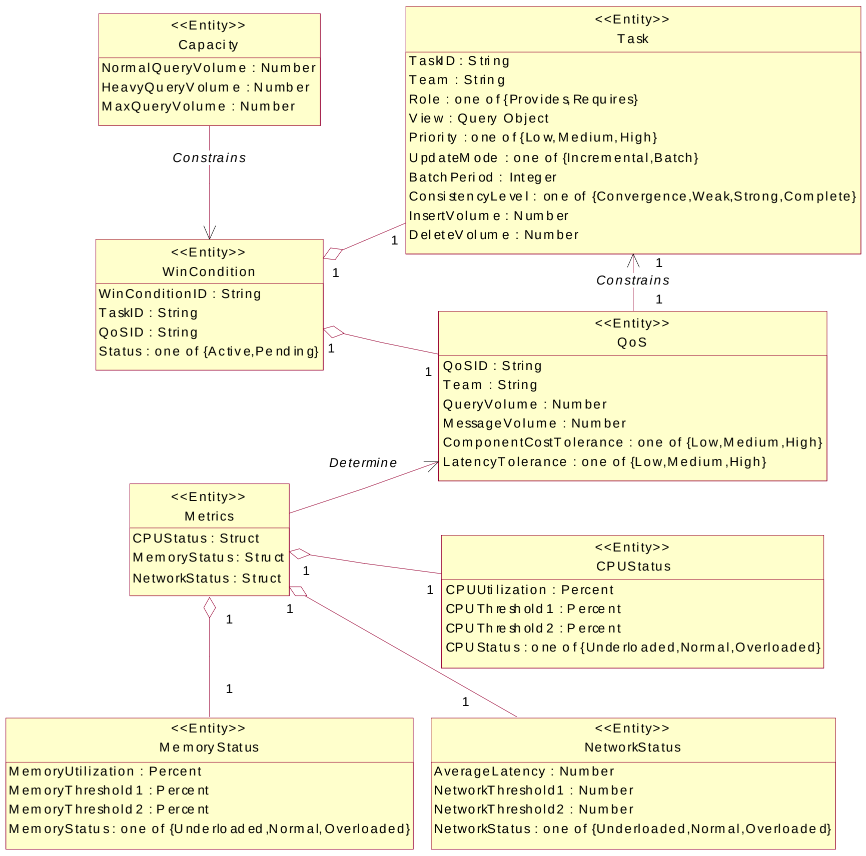
```
1. CNA submits a WinCondition to NFA. The WinCondition identifies the task
preferences and constraints of the Client
2. The NFA analyzes the posted WinCondition and identifies Issue(s)
3. The NFA generates potential Options that Resolve the Issue(s): Options are policy
decisions that are either derived from a) the resource (RNA) preferences or b) global
policy knowledge
4.  NAs (both CNA and RNAs) evaluate the Option(s)
5.  If  an option is accepted by all NAs
    Then {Agreement = Accepted Option, Agreement propagated to CCA for
implementation}
   Else {one or more NAs post revised WinConditions
           Go To Step 2 }    End If
```

Table 4. Conflict Driven Negotiation Protocol

```
1.  CNA or RNA submits revised WinCondition to the NFA.
2. NFA analyzes revised WinCondition against existing related WinConditions to
generate Issue(s) resulting from conflicting interaction
3. NFA generates potential Options that Resolve the Issue(s)
4.  If no change in existing Options (i.e., Option has already been Agreed upon)
    Then {NFA marks the issue as Resolved}
     Else { CNA and RNAs evaluate the Option(s)
.             If  an option is accepted by all NAs
              Then {Agreement = Accepted Option, Agreement propagated to CCA}
              Else {CNA and/or RNAs post revised WinConditions, Go To Step 2 }
              End If
      End If
```

The third method is priority driven and is used when an acceptable policy cannot be negotiated among all team participants in a predetermined amount of time. In this scheme, team participants are assigned a priority based on inputs from the task owner. The option with the highest overall utility, based on global and team member priorities, is selected.

The above negotiation reasoning methods exploit the context and view maintenance problem domain to generate the issues and options and to evaluate the options as follows: 1) Given one or more WinConditions, *issue generation* involves formulating a query to identify VMP specification objects that satisfy the WinConditions. Here the issue is formalized as a query object (global goal generation). 2) Given the formulation of the issue, *option generation* involves evaluation of the query to retrieve plausible VMP specification objects and their refinements based on action-theory knowledge of the NFA. 3) Given the options, *option evaluation* involves checking for consistency of an option against the committed WinConditions representing active beliefs of the RNAs.

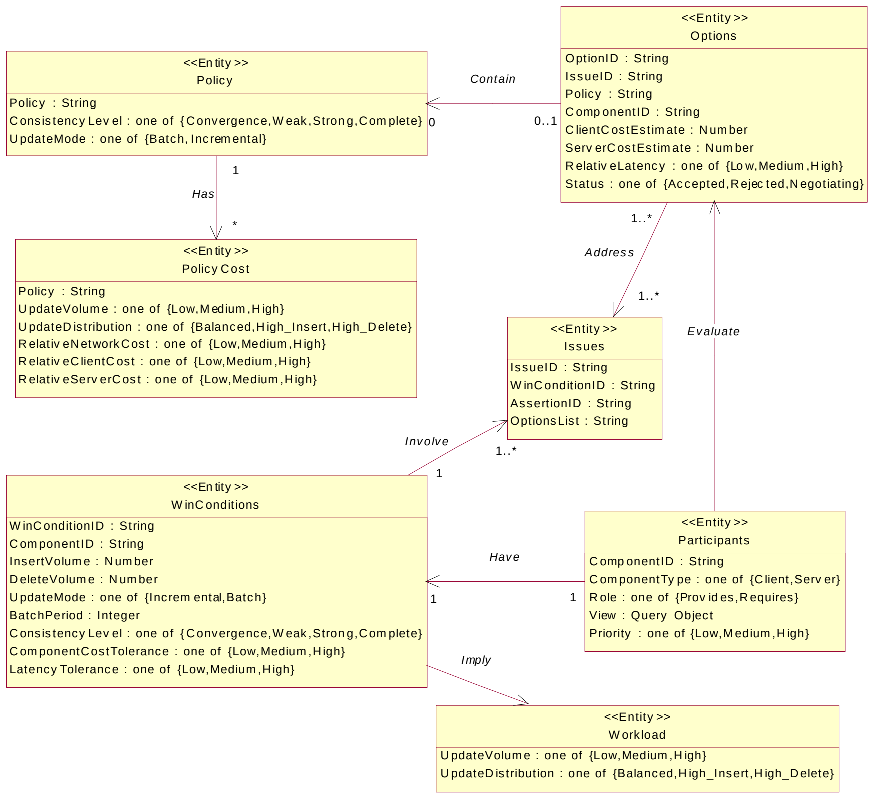


<<Entity>>
Capacity
NormalQueryVolume : Number
HeavyQueryVolume : Number
MaxQueryVolume : Number
Constrains
<<Entity>>
Task
TaskID : String
Team : String
Role : one of {Provides, Requires}
View : Query Object
Priority : one of {Low, Medium, High}
UpdateMode : one of {Incremental,Batch}
BatchPeriod : Integer
ConsistencyLevel : one of {Convergence,Weak,Strong,Complete}
InsertVolume : Number
DeleteVolume : Number
<<Entity>>
WinCondition
WinConditionID : String
TaskID : String
QoSID : String
Status : one of {Active,Pending}
1
1
1
Constrains
1
1
<<Entity>>
QoS
QoSID : String
Team : String
QueryVolume : Number
MessageVolume : Number
ComponentCostTolerance : one of {Low,Medium,High}
LatencyTolerance : one of {Low,Medium,High}
1
1
Determine
<<Entity>>
Metrics
CPUStatus : Struct
MemoryStatus : Struct
NetworkStatus : Struct
1
1
1
1
1
1
<<Entity>>
CPUStatus
CPUUtilization : Percent
CPUThreshold1 : Percent
CPUThreshold2 : Percent
CPUStatus : one of {Underloaded,Normal,Overloaded}
<<Entity>>
MemoryStatus
MemoryUtilization : Percent
MemoryThreshold1 : Percent
MemoryThreshold2 : Percent
MemoryStatus : one of {Underloaded,Normal,Overloaded}
<<Entity>>
NetworkStatus
AverageLatency : Number
NetworkThreshold1 : Number
NetworkThreshold2 : Number
NetworkStatus : one of {Underloaded,Normal,Overloaded}

Fig. 4. RNA and CNA Data Model

4.1 Models to Support Negotiated Selection of VMP

In order to support the agent negotiation coordination protocols outlined above, NAVCo defines the client and resource negotiating agents to have a) declarative models of preferences and constraints represented as a database of facts, and b) action-theory for issue generation, option generation and evaluation that are represented as a set of rules. We briefly describe below the data models and some examples of the rules that have been formulated and prototyped in our initial experiments.



<<Entity>>
Policy
Policy : String
ConsistencyLevel : one of {Convergence,Weak,Strong,Complete}
UpdateMode : one of {Batch, Incremental}
Contain
0
0..1
<<Entity>>
Options
OptionID : String
IssueID : String
Policy : String
ComponentID : String
ClientCostEstimate : Number
ServerCostEstimate : Number
RelativeLatency : one of {Low,Medium,High}
Status : one of {Accepted,Rejected,Negotiating}
1
Has
*
<<Entity>>
PolicyCost
Policy : String
UpdateVolume : one of {Low,Medium,High}
UpdateDistribution : one of {Balanced,High_Insert,High_Delete}
RelativeNetworkCost : one of {Low,Medium,High}
RelativeClientCost : one of {Low,Medium,High}
RelativeServerCost : one of {Low,Medium,High}
1..*
Address
1..*
Evaluate
<<Entity>>
Issues
IssueID : String
WinConditionID : String
AssertionID : String
OptionsList : String
Involve
1..*
1
<<Entity>>
WinConditions
WinConditionID : String
ComponentID : String
InsertVolume : Number
DeleteVolume : Number
UpdateMode : one of {Incremental,Batch}
BatchPeriod : Integer
ConsistencyLevel : one of {Convergence,Weak,Strong,Complete}
ComponentCostTolerance : one of {Low,Medium,High}
LatencyTolerance : one of {Low,Medium,High}
Have
1
1
<<Entity>>
Participants
ComponentID : String
ComponentType : one of {Client,Server}
Role : one of {Provides,Requires}
View : Query Object
Priority : one of {Low,Medium,High}
Imply
<<Entity>>
Workload
UpdateVolume : one of {Low,Medium,High}
UpdateDistribution : one of {Balanced,High_Insert,High_Delete}

Fig. 5. NFA Data Model

The class diagram shown in Figure 4 captures the data model underlying the information maintained by the CNA of the clients and the RNA of the resources. The model in essence articulates the WinCondition as consisting of two parts: a) Task part of type provides or requires depending on the component type (resource or client). The task part explicates the role to be played, prioritization of tasks, task preferences, and update volume and distribution submitted to the team in support of the task. b) QoS constraint part articulating the constraints imposed on the task. The QoS schema specifies the component workload to support the task and the component QoS constraints based on the status of component resources captured as QoS metrics. The data model also specifies global integrity constraints in terms of the capacity of the clients and resources.

The data model specifying the content of the information in the NFA is given in Figure 5. The data-model captures specifications of the VMPs and their associated costs. The data model also contains models of the WinConditions, Issues, and Options that get posted to or generated by the NFA. Some of the important data elements are a) identification, characteristics, and costs of available coordination policies, b) task-specific meta-data, and c) overall team-level workload characterization, preferences, and constraints.

The rules for issue and option generation are modeled as database trigger rules that analyze an update and create issues, options and option evaluations. The trigger rule in Table 5 creates an Issue, whose semantics is that of a query assertion to select a Policy, in the Issues tables when there is an update in the WinCondition table. It accesses relevant constraints imposed by a task specific WinCondition that must be met by a VMP.

Table 5. An Example of an Issue Generation Rule modeled as a Trigger Rule

```
TRIGGER <Issue generation> on INSERT into WinConditions
(INSERT into Issues (…)
    WHERE Issue.Assertion =
   (SELECT Policy
     FROM PolicyCost |x| Policy
WHERE UpdateVolume = WinCondition.UpdateVolume
AND UpdateDistribution =WinCondition.UpdateDistribution
AND ConsistencyLevel = WinCondition.ConsistencyLevel
AND UpdateMode=WinCondition.UpdateMode
AND RelativeClientCost < =WinCondition.ComponentCostTolerance))
```

5 Design and Analysis of Change Coordination Layer

A major requirement on the agent-based mechanisms for run-time change coordination of view maintenance policies is ensuring *application state independence.* This involves ensuring that the sequence of change actions, imposed by the change agents, leads to a connection state that is consistent with the application level processing state without the policy change. The key idea underlying the NAVCo approach to addressing the above requirement is based on the understanding that ensuring such a property involves a) use of knowledge of the processing state of the current view maintenance policy components, b) exploiting knowledge of the behavior of the new policy component to identify the starting state of the new policy such that continued processing from that state would be a consistent progress of the processing state of the current policy, and c) a sequence of actions that brings about the change in activities of the policies and their respective states.

The above understanding is translated into agent design constraints by having 1) tracking agents that use abstract specifications of the view maintenance policies to track their processing states, and 2) a change agent to coordinate the actions required to bring about the change.

The design of the change agents is based on decision rules for control and configuration of the policy-based application layer coordination agents. The rules are obtained by systematic pairwise analysis of switching from one policy to another and identifying changes that do not introduce any inconsistencies in the application specific processing states (such as inconsistent workloads). The questions then are: 1) What are the right abstract behavioral specifications of the view maintenance policy agents that need to be tracked? 2) What is the method for obtaining the specifications of the change agents? 3) Given the error-prone manual generation of the decision rules that get used to program the change agents, how do we analyze and debug the correctness of the decision rules used by the change agents? The following two subsections describe the NAVCo approach to specifying the agent-behaviors for tracking and generation of the rule-based specifications of the change agents.

5.1 Tracking Agent: Behavior Specification

The answer to the first question, raised above, is based on domain analysis of the algorithms for view maintenance. In particular, we identify the state variables underlying the scheduling agent, the query processing agent and the input buffer that represent the application specific processing state and are manipulated by the algorithms to provide view maintenance. Based on such analysis, we have identified the following set of explicit state variables represented by the different components: a) the scheduler's current workload defining the incoming updates from the application specific information resources, b) the query processor's pending tasks (called here the unanswered query set), c) the tasks that have been completed but yet to be scheduled for propagation into the client view, and d) the concurrent update tasks that arrive at the buffer during the processing of an update (necessary to identify and execute compensating queries to eliminate the inconsistency of the current answer due to interacting concurrent updates).

For a given view maintenance policy, the abstract behavior specification is based on qualitative abstractions of the state variables. The abstractions aid in characterizing the overall workload of the different components in the design for a specific policy, that must be gracefully preserved during transitions between view maintenance policies in order to satisfy the application state independence criteria.

5.2 Change Agent: Behavior Specification

The behavior specification of the Change Agent is modeled as a decision tree captured by a finite state machine model (activity diagram in UML) where the decision conditions are captured by the transitions and the actions are captured by the activity states. The decision tree is obtained by a two-step process.

Step 1 - Identify intermediate states. In this step, different workload scenarios are considered and the policies are manually executed to generate event traces for each policy against each workload scenario.

Figure 6 shows a scenario consisting of a single insert operation with no concurrent updates. The scenario modeled as a sequence diagram shows a partial interaction between the application components and the view maintenance agents, where: 1) An insert is submitted by the RMA of an information resource and received by the active scheduling agent (SA). 2) The SA generates a query based on the received insert and schedules it for execution by the query processing agent (QPA). 3) The QPA decomposes the query and sends sub-queries to other resources as required to generate an incremental view update. 4) The QPA sends the query answer to the SA. 5) The SA processes the update and propagates it to the CMA of the client view.

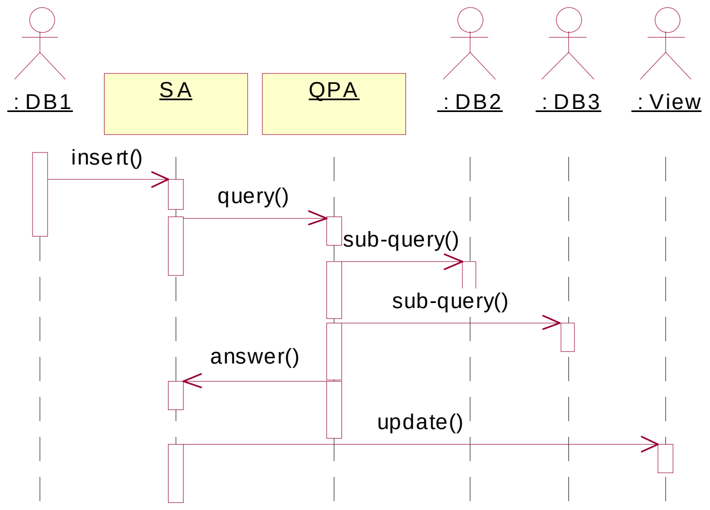


Fig. 6. An example workload scenario

Table 6 shows the event trace table generated by C-Strobe based on the workload scenario in Figure 6. The table shows the sequence of the distinct abstract state vectors (corresponding to the qualitative abstractions of the state variables identified in the previous sections and tracked by the tracking agents in the architecture) that result from the execution of the view maintenance objects. Similarly, Table 7 shows the event trace table for the Strobe policy objects.

Step 2 - Obtain change plans. The second step involves analyzing for pairwise switching of policies to identify workload constraints of the active policy in a given state (specified by the abstract state vector) in the trace and the matching workload constraints of the new policy to be activated in a specific state. The configuration action for activating the new policy is then setting up the workload state variables according to the valid transitions obtained from the above analysis.

This step consists of two sub-steps: 1) identifying the entry in the second trace such that continued processing from that point in the trace would be a consistent progress of the processing state of the first trace, and 2) determining the sequence of actions required to map the state variable values of the first trace to appropriate state variable values of the second trace.

Table 6. Partial event trace generated by C-Strobe

ID	Event Sequence	WL	UQS	Delta	CD	State
1	initial (idle)	Z	E	E	E	W
2	WL=WL+U1	**I**	E	E	E	**T**
3	Q1,1,0=V(U1)	**I**	E	E	E	EV
4	UQS=UQS+Q1,1,0	**I**	**N**	E	E	EV
5	**A1=source_evaluate(Q1) to QP**	**I**	**N**	E	E	EV
6	**processAnswer(A1) from QP**	**I**	**N**	E	E	**T**
7	Delta=Delta+A1,1,0	**I**	**N**	**N**	E	**T**
8	UQS=UQS-Q1,1,0	**I**	**E**	**N**	E	**T**
9	CD=checkBuffer()	**I**	**E**	**N**	E	P
10	Delta=null	**Z**	**E**	**E**	E	**U**
11	WL=null	**Z**	**E**	**N**	E	**T**
12	final (idle)	**Z**	**E**	**E**	E	W

Table 7. Partial event trace generated by Strobe

ID	Event Sequence	WL	UQS	AL	PL	STATE
1	initial (idle)	0	E	E	E	W
2	WL=WL+U1	**>0**	E	E	E	**T**
3	Q1=V(U1)	>0	E	E	E	EV
4	UQS=UQS+Q1	>0	**NE/FP**	E	E	EV
5	set pending(Q1)=null	>0	NE/FP	E	**NE**	EV
6	**A1=source_eval(Q1) to QP**	>0	NE/FP	E	NE	EV
7	**processAnswer(A1) from QP**	>0	NE/FP	E	NE	**T**
8	UQS=UQS-Q1	>0	**E**	E	NE	**T**
9	interim state	>0	E	E	NE	P
10	delete pendingList(Q1)	>0	E	E	**E**	P
11	AL=AL+A1	>0	E	**NE**	E	P
12	AL=null	>0	E	**E**	E	**T**
13	WL=null	**0**	E	E	E	**T**
14	final (idle)	0	E	E	E	W

6 Prototype

The adaptive view coordination architecture has been modeled using Rationale Rose 98 Enterprise Edition. Use cases, class diagrams, object collaboration diagrams, and sequence diagrams have been developed. Initial prototypes have been developed for both the negotiation and application views (layers). Prototypes for resource manager, resource negotiation, change coordinator, and negotiation facilitator agents have been developed. Each prototype agent consists of a Java application and a Microsoft Access database.

All agent-to-agent coordination is accomplished through the use of the Nspace, which is implemented using JavaSpaces technology. WinConditions, options, dynamic switching plans and other objects are written as entries into the Nspace. The Nspace notify and read methods are utilized to route the entries to the appropriate agents. The prototype agents currently utilize input and output text files to simulate interactions with clients and resources. Initial results show that the NAVCo reactive reasoning methods can exploit the JavaSpaces based design environment to make negotiated decisions on the policy objects.

7 Related Work

There has been a significant amount of work conducted in the area of view maintenance resulting in a spectrum of solutions ranging from a fully virtual approach where no data is materialized at one extreme to a fully replicated approach where full base relations are copied at the other extreme. The Strobe [21] and SWEEP algorithms [1] are a hybrid of these two extremes and are designed to provide incremental view maintenance over multiple, distributed resources.

The NAVCo work builds on the negotiation research performed by the community in requirements negotiation as well as automated negotiation. Negotiation is a complex and difficult area of active research pursued by researchers in different fields of study. Research progress has been made in different approaches to negotiation: a) Human Factors approach - here the major focus is understanding methods and techniques employed by humans to negotiate so as to manage the human factors of pride, ego, and culture [8, 15, 16]. The work on understanding people factors in requirements negotiation falls in this category. b) Economics, Game Theory and Bargaining approach - here research progress has been made on theoretical models of outcome driven negotiation and self-stabilizing agreement to achieve some equilibrium [11] and process driven negotiation [14]. Research on negotiation focuses on the group decision context where the power to decide is distributed across more than one stakeholder/agent as opposed to group decision making where a single decision maker relies on a set of analysts [12]. Two key aspects of the negotiated decision studied in most of the research are conflict and interdependence of decisions. Conflict has been used constructively in cooperative domains to explore negotiation options [3]. c) Artificial Agents approach - here the focus has been on developing computational agents that negotiate to resolve conflict [6], to distribute tasks [17, 19], to share resources [22], to change goals so as to optimize multi-attribute utility functions [18]. In general, the models for agent cooperation and negotiation consider negotiation between multiple agents driven by global utility functions or driven by independent local utility functions. The WinWin [2, 3] model used in NAVCo considers both types of drivers typical of negotiating teams having local preferences as well as global constraints.

The NAVCo approach is also similar in spirit to the work on architecture-based run-time evolution [13]. Our approach and reasoning tools differ from [13] in terms of the nature of automation. The work in [13] focused on providing a support environment where the necessary analysis for dynamic change and consequent operationalization can be performed. The NAVCo approach and prototype discussed in the paper is motivated by automated switching via automated negotiation reasoning and change coordination realized by the middleware agents in the negotiation layer and the change coordination layer.

8 Summary and Future Work

This paper develops a Negotiation-based Adaptive View Coordination (NAVCo) approach for a class of distributed information management systems that allows view maintenance policies to be dynamically adapted to meet changes in QoS preferences and constraints in a run-time environment. The key ideas in the NAVCo approach involve negotiation-based coordination and model-based tracking and policy change coordination. The ideas are well suited for realization using agent-based concepts and architecture. The paper describes the layered agent-based architecture with specific emphasis on the agent-oriented middleware at each layer that supports a) cooperative change reasoning via multi-agent negotiation coordination and negotiated artifacts representations, and b) change coordination via model-based tracking and change agents. With respect to the change-coordination layer, the paper details systematic approaches to specifying the tracking agents and synthesizing the rule-based specifications of the change agents. Current work is targeted towards developing monitoring agents that monitor the application layer middleware agents and can be used to trigger the policy switching process based on reifying the run-time performance concern as a change in committed WinConditions. Also such monitors would be useful to validate the adaptation in terms of performance benefits gained as a result of switching. Future work will focus on debugging the agent specifications using model checking approaches as well as further extensions to the capability of the negotiating agents for issue generation, option generation and evaluation rules.

9 References

1. D. Agrawal, A. El Abbadi, A Singh, and T. Yurek, Efficient View Maintenance at Data Warehouses. In Proceedings of the ACM SIGMOD ‘97, pp. 417-427, 1997.
2. B. Boehm, P. Bose, E Horowitz and M. J. Lee, “Software Requirements Negotiation and Renegotiation Aids: A Theory-W Based Spiral Approach,” In IEEE Proceedings of the 17th ICSE Conference, 1995.
3. P. Bose and Z. Xiaoqing, "WinWin Decision Model Based Coordination,” International Joint Conference on Work Activities Coordination and Collaboration, 1999.
4. P. Bose and M. G. Matthews, “An Architecture for Achieving Dynamic Change in Coordination Policies,” Proceedings of the 4th International Software Architecture Workshop (ISAW4), June 2000.
5. W. R. Collins et. al., "How Good is Good Enough?" Communications of the ACM, pp. 81-91, January 1994.
6. E. H. Durfee, V. R. Lesser, and D. D. Corkill, "Cooperation Through Communication in a Distributed Problem Solving Network,” In M. N. Huhns ed., Distributed Artificial Intelligence, Chapter 2, 29-58.

7. J. Farley, "Java Distributed Computing," O' Reilly Press, 1998.
8. R. Fisher and W. Ury, "Getting to Yes," Houghton-Mifflin, 1981. Also Penguin Books, 1983.
9. R. Hull and G. Zhou, A Framework for Supporting Data Integration Using the Materialized and Virtual Approaches. In Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 481-492, June 1996.
10. T. W. Malone and K. Crowston, "The Interdisciplinary Study of Coordination," ACM Computing Surveys, Vol. 26, No. 1, pp. 87-119, Mar. 1994.
11. J. F. Nash, "The Bargaining Problem," Econometrica 28, pp. 155-162, 1950.
12. J. F. Nunamaker, A. R. Dennis, J. S. Valacich and D. R. Vogel, "Information Technology for Negotiating Groups: Generating Options for Mutual Gain," Management Science, October 1991.
13. P. Oreizy, N. Medvidovic, and R. N. Taylor, "Architecture-based Runtime Evolution," In Proceedings of ICSE 1998.
14. M. J. Osborne and A. Rubinstein, "A Course in Game Theory," MIT Press, MA, 1994.
15. M. Porter, "Competitive Strategy: Techniques for Analyzing Industries and Competitors," Free Press, NY, 1980.
16. H. Raiffa, "The Art and Science of Negotiation,, Harvard University Press, Cambridge, MA, 1982
17. R. G. Smith, "The Contract Net Protocol: High-level Communication and Control in a Distributed Problem Solver," IEEE Trans. On Computer, 29, pp. 1104-1113, 1980.
18. K. P. Sycara, "Resolving Goal Conflicts Via Negotiation," In Proceedings of AAAI, pp. 245-250, 1988.
19. M. P. Wellman, "A General Equilibrium Approach to Distributed Transportation Planning," In Proceedings of AAAI-92, San Jose, CA 1992.
20. T. Winograd, "Bringing Design to Software," Addison Wesley Publishers, 1996.
21. Y. Zhuge, H. Garcia-Molina, and J. Wiener, "The Strobe Algorithms for Multi-Source Warehouse Consistency," In Proceedings of the International Conference on Parallel and Distributed Information Systems, December 1996.
22. G. Zlotkin, and J. S. Rosenschein, "Mechanism Design for Automated Negotiation and Its Application to Task Oriented Domains," Artificial Intelligence, Vol. 86, pp. 195-244, 1996.