STATE BASED MODELING OF SCHEDULING AGENTS IN INTELLIGENT MANUFACTURING

by

Kamer Sözer

B.S. in Mechanical Engineering, Boğaziçi University, 2004

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Industrial Engineering
Boğaziçi University

2007

STATE BASED MODELING OF SCHEDULING AGENTS IN INTELLIGENT MANUFACTURING

APPROVED BY:

Assoc. Prof. Ali Tamer Ünal (Thesis Supervisor)	.....
Prof. H. Levent Akın	.....
Prof. Ümit Bilge	.....

DATE OF APPROVAL: 26.01.2007

ACKNOWLEDGEMENTS

I would like to express my grateful and sincere thanks to my supervisor, Assoc. Prof. Ali Tamer Ünal for his encouraging and guidance throughout the realization of this thesis. His wide knowledge, constructive comments and merry intellectual conversations have been great value for me. The inspiration that he provided me will be my support for all my life.

I would like to thank Prof. Ümit Bilge and Prof. H. Levent Akın for their helpful suggestions and comments.

I also wish to extend many thanks to all my friends who have consistently helped and supported me during the preparation of this thesis.

Finally, I would like to thank to my family for their moral support.

ABSTRACT

STATE BASED MODELING OF SCHEDULING AGENTS IN INTELLIGENT MANUFACTURING

In dynamic and stochastic manufacturing environments, a scheduling system should be able to modify its schedule to adapt to changing situations such as machine breakdowns, due-date changes, new rush orders, canceled orders and so on. In centralized monolithic scheduling systems, dynamic updating may be prohibitive mostly due to computational constraints in generating new schedules, or due to non-existence of effective shop floor data monitoring/data collection systems across the organization. One of the approaches to generate timely scheduling decisions, especially for large scale problems, is to decompose the problem and distribute responsibility of sub-problems to various agents. When agents are organized as loosely coupled autonomous scheduling units, it is easier to manage and control the data with respective local availability constraints, and eventually, faster to respond to system changes. However, distributing the scheduling problem brings in additional issues such as: choice of the decomposition methodology (defines the boundaries of independence for the agents); accuracy and timing of representation of local reality in each agent; scheduling algorithms and frequency of re-scheduling in each agent; and maybe most importantly, handling mechanism for coupling constraints between agents. In this study, we propose a state based modeling approach to represent the workings of a scheduling agent so that effect of choices made in distributing the problem in terms of the overall performance of the scheduling methodology can formally be defined and analyzed. The proposed model will be a framework to properly identify conditions and timing of possible inconsistencies and inaccuracy, and analyze the effectiveness of various distribution mechanisms in terms of scheduling performance.

ÖZET

AKILLI İMALATTAKİ ÇİZELGELEME ARACININ DURUM TABANLI MODELLEMESİ

Dinamik ve stokastik üretim ortamlarında, bir çizelgeleme sistemi, oluşturduğu çizelgeyi makina bozulmaları, vade tarihi değişimleri, yeni acil siparişler, feshedilmiş siparişler ve bunun gibi değişikliklere göre uyarlama yeteneğine sahip olmalıdır. Merkezi ve tek para çizelgeleme sistemlerinde, dinamik uyarlamalar, yeni çizelgeler türetmedeki hesaplama zorlukları veya tüm şirket satında etkin atölye veri izleme / veri toplama sistemlerinin yokluğu sebebiyle genelde engellenirler. Uygun çizelgeleme kararları türetmek için varolan yaklaşımlardan biri, özellikle geniş çaplı problemler için, problemi bölmek ve alt problemlerin sorumluluğunu birkaç araca dağıtmaktır. Aracılar gevşek bağlı özerk çizelgeleme birimleri şeklinde düzenlendiklerinde, veriyi yerel kullanılırlık kısıtlarına göre yönetmek ve kontrol etmek daha kolaydır, ve nihayet, sistem değişikliklerine yanıt vermek daha hızlıdır. Ancak, çizelgeleme probleminin dağıtılması ilave sorunlar arz eder: Bunlar, Parçalama yönteminin seçimi (ki bu aracıların bağımsızlığının sınırlarını belirler); Her aracıdaki yerel gerçekliğin temsilinin doğruluğu ve zamanlaması; Her aracıdaki çizelgeleme algoritması ve tekrar çizelgeleme sıklığı; ve belkide en önemlisi Aracılar arasındaki bağlaşım kısıtları için çözüm mekanizmaları sorunlarıdır. Bu çalışmada, çizelgeleme aracılarının çalışmalarını tasvir edebilmek icin durum tabanlı bir modelleme yaklaşımı öneriyoruz, böylece problemin dağıtımında yapılan seçimlerin etkisi, çizelgeleme yönteminin genel başarımına dayalı olarak, resmen tanımlanabilecek ve incelenebilecektir. Önerilen model, olası tutarsızlıkların ve kusurların zamanlamasını ve koşullarını gerektiği gibi teşhis etmek için ve değişik dağıtım mekanizmalarının etkililiğini çizelgeleme performansına dayalı olarak incelemek için bir çatı olacaktır.

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

Scheduling is the determination of operation start and completion times on resources in order to achieve some objectives while taking customer requirements and system constraints into consideration. Scheduling problems are hard to solve theoretically and for most of these problems are NP-complete and it not possible to find the optimal solution to real life problems in a reasonable amount of time.

Characteristics of today's manufacturing environment are small to mid-sized production batch sizes, short production cycles, high value to customer satisfaction which means to meet the requirements of tight due dates. Thus timely dispatching of schedules is a challenge in such environments in the face of disruptions, production parameters that are prone to variability, and changes in order information.

How scheduling process works is as follows: the production problem in real life is modeled for the software environment; the related data is collected from the shop floor and enterprise data stores, then the instances of the problem are represented in this decision making environment. After the representation of the problem, the appropriate scheduling algorithm is run. Then available plan is dispatched to the shop floor.

Solution methods for scheduling problem are grouped into two: Optimal Methods and Heuristic Methods. Branch and Bound, Dynamic Programming, Mathematical Programming are optimal methods, and they work on small sized problems, so they are not applicable for most of the production scheduling problems. On the other hand heuristic methods contain Dispatching Rule Based Methods (such as SPT, EDD, Minimum Slack, etc.), Neighborhood Search Methods (such as Simulated Annealing, Genetic Algorithms, TABU Search etc.) and Constraint Programming. Heuristic methods can be evaluated by determining worst case bound among all instances or by calculating average deviation from the optimal solution. In general the performance of solution methods highly depend on structure of the problem and the way the parameters are defined, i.e. the model built for the problem.

There are also Repair Methods that are called Dynamically Updated Heuristics. After a schedule is constructed in some way, it is evaluated, and then certain modifications are made to repair it. Repair methods are very important for production scheduling in dynamic and stochastic environments, because in scheduling problems, a little disturbance in the problem instance results large deviations in the solution, thus modifications or corrections are required. In other words, even slight change of parameters may cause the solution at hand not to work for the new problem, then repair methods are applied to adjust the parameters for the current problem. In application, one useful approach is to collect data at some moment in time and freeze it (i.e. a "snapshot" of the data is taken) then the optimal schedule for the data on hand would be determined and used until the next snapshot is taken (Lawrence and Sewell, 1997). In particular, dynamically updated heuristic schedules may perform better than static optimal sequences when uncertainties about the production environment are introduced into the model (Lawrence and Sewell, 1997). In general, dynamic and instance dependent corrections may provide a simple dispatching performance algorithm performs well (Kutanoğlu and Sabuncuoğlu, 2001), since these corrections try to capture the current situation of the problem.

However, in most of the implementations, dynamic updating can not be performed due to computational constraints in generating new schedules, or lack of scheduling knowledge or non-existence of effective shop floor data monitoring/collection systems (Simsek, 2000). More precisely, in some cases scheduling process takes too much time, and when the scheduling process of an instance is finished, that instance will probably be changed and thus, does not exist with the initially given conditions anymore. Moreover, the real world scheduling problems are almost always much complex and uncertain than the models that can be built to represent the problem (Lawrence and Sewell, 1997). When a number of events including machine breakdowns, due-date changes, new rush orders, canceled orders and so on, introduced to the model, the model gets complicated. These events necessitate a reasonable modification of the schedule during the construction of the schedule, so rapid and operative data monitoring/collection systems are required.

Researchers try to develop effective and more practical scheduling techniques with less computational requirements for use in production problems. One of the approaches to generate timely scheduling decisions, especially for large scale problems, is to decompose the problem and distribute to various servers/agents. This approach is called distributed scheduling (Oğuz, 1996). Multi-agent systems have been studied for many years and various types of such systems have been developed. Examples include distributed artificial intelligence (DAI) systems, agent-based manufacturing system (ABMS) and coordination protocols (Anussornnitisarn and Nof, 2005). Common concept in these systems is that groups of similar or complimentary agents act together to solve problems based on their value systems and communication abilities (Gou et al., 1998).

From the distributed scheduling perspective, scheduling problem is decomposed into sub-problems; then the responsibility of scheduling decisions is shared by local decision makers which are called agents. In this study, we assume that scheduling agents operate as follows: Agents load data or part of the data about the instance of the problem they solve to their individual domains, then they make scheduling decisions by their individual computing facilities based on their local information and local decision-making protocol (scheduling algorithms, and scheduling triggers).

When agents are organized as loosely coupled autonomous scheduling units, it is easier to manage and control the data and faster to respond the system changes. Thereby a very complex scheduling problem can be reduced to some tractable cases where exact or approximate algorithms can be applied for their efficient solution(Simsek, 2000); whereas it can be difficult with a centralized scheduling approach to achieve objectives in highly dynamic systems due to computational constraints in generating new schedules.

In dynamic problems for reactive scheduling the main issue is to determine “when to schedule/reschedule”, however, application of a distributed scheduling approach to the problem brings in additional complexity because the system is not purely separable. More precisely coupling constraints in the face of decomposition bring additional issues

to be resolved. The first one is "how to decompose". There are various methods for the decomposition of the production problem. Some of them propose machine based decomposition and face with coupling constraints resulting from jobs. Similarly in job based decomposition, machine constraints are introduced as coupling constraints. In either case there are a set of scheduling problems solved by different scheduling agents that have coupling constraints among them; this can lead to inconsistencies (Anussornnitisarn and Nof, 2005), for instance processor-successor relations among operations of a job is violated. Thus another issue is "how to resolve the system conflicts". System conflicts may result from the following:

Consistency of the represented data; The agents are supplied by the information or the part of the information about the instance of the problem they solve. The data is represented to their individual domains, then agents make scheduling decisions by their individual computing facilities based on their local information. Representation of the current situation to the agent's local domain is a hard job in dynamic, complex and large systems. Because the representation process (i.e reading the data and transforming it to the respective domain) takes time and when it is finished, that instance will probably be changed and thus, does not exist with the initially given conditions anymore.

The reason of the change in the instance of the data may be due to dynamic nature of the environment likewise existence of another processes related to the same data. In order to provide "consistent representation", simultaneous access of multiple processes may be prevented. This is a synchronization issue and there are related mutual exclusion algorithms to ensure a shared resource is accessed by at most one process at a time (Tanenbaum and Steen, 2001). To make the statement clear, suppose that some data is required by an agent, mutual exclusion algorithms provide that when representation process starts any other processes which will transform the mentioned data or part of the data is blocked. However it may be the case that actual situation is changed but necessary transformation to signify the change is blocked because no transformation is allowed on mentioned data until the representation process finishes. As a result, the representation in local domain is now consistent but not valid.

Consistency of the shared data among related agents; As mentioned above, since the problem is not purely separable, coupling constraints are introduced to the model in the face of decomposition of the problem. Coupling constraints are presented as the shared data to related scheduling agents. Consistency of the shared data among the agents is failed in case when dissimilar representations of the corresponding instance occur. The scenario is as follows: shared data is supplied to one of the local domains then, a change occurs and the shared data is transformed before it is supplied to other related local domains. This is another synchronization issue that multiple processes may sometimes need to agree on the ordering of the events. In distributed systems it is a considerable problem to provide relative ordering whereas it does not matter in centralized systems (Tanenbaum and Steen, 2001). In order to handle this kind of situations, an approach can be modified. It is about event ordering problem and is an extension of Lamport's algorithm, which synchronizes logical clocks called vector timestamps (Tanenbaum and Steen, 2001). By the Lamport's algorithm all system messages are universally ordered into a sequence but now the system acts like as it is centralized (i.e. single thread).

Conflicts due to different scheduling policies among related scheduling agent; Last but not least, there may be conflicts even when the same instance of the shared data could be represented to the related agents domain. In other words, perfect synchronization is almost impossible to achieve but even it were not the case, there may be still system conflicts because of the individual and different solution policies applied by the related scheduling agents which have coupling constraints among them. Agents make scheduling decisions by their individual computing facilities based on their local information, and different policies result along with different solutions for the same problem.

In order to resolve the conflicts of the system, coordination, cooperation, and communication mechanisms are proposed in the literature. These negotiation mechanisms try to synchronize all the units in the system to achieve the same common goal i.e. globally consistent system. However negotiation mechanisms mostly force the system to act as it is centralized besides they cause the system to slow down. Moreover,

in accordance with the goal of consistency, synchronization mechanisms, in particular, locking mechanisms are introduced. But the purpose of distributing is violated if the entire system gets stopped waiting for a lock to be released. Here comes the question that "Are feasibility and consistency worth to be sustained when seeking for an effective solution method to the distributed scheduling problem?".

The motivation behind this study is that some obstacles that may lead the system to become slower, may easily be recognized when the system gets easily observable by providing effective shop floor data monitoring system. Instead of making an effort for eventually consistent system, some conflicts may be allowed or some feasibility constraints may be violated in a controlled environment. From the control point of view, there should exist some parameters worth to be tuned in order to make effective decisions against system variability. The aim is to propose an effective scheduling mechanism by improving the control tools and managing the adjustment of essential parameters based on the current status of the entire system.

In this study, we propose a state based modeling approach to represent the workings of a scheduling agent so that effect of choices made in distributing the problem in terms of the overall performance of the scheduling methodology is formally be defined and analyzed. In "Problem Definition" chapter, the problem is defined and expressed in mathematical terms, thus we mathematically define possible states of objects in the system within the scheduling process. In "Model" chapter, the proposed model will be explained conceptually in detail. In "ICRON Implementation" chapter, the ICRON software is introduced and implementations in this software are explained, and finally before the conclusion, explanation of the Distributed Simulator is provided. Distributed Simulator is developed in order to identify conditions and timing of possible inconsistencies and inaccuracy, and analyze the effectiveness of various distribution mechanisms in terms of scheduling performance.

2. LITERATURE REVIEW

The Literature review is mainly focused on: “Dynamic Scheduling”, “Negotiation Mechanisms for Distributed Systems”, and “Distributed Modeling Techniques”.

2.1. Dynamic Scheduling

Lawrence and Sewell (1997) discussed the utility of the optimizing solution methodologies to solve practical job-shop production scheduling problems. They searched the answer of the question that ”are optimal solution techniques worth the developmental and computational effort when practical scheduling problems are almost always more complex and uncertain than models can be build?”. They examined the job-shop problem with make span objective when job processing times are uncertain. In order to make comparison of optimal and heuristic schedules, the methodology is to generate optimal, near optimal, and heuristic sequences for a variety of problems using expected processing times, and then to compare these sequences when processing times are randomly varied. Results indicate that fixed optimal sequences derived from deterministic assumptions get worse with the introduction of processing time uncertainty when compared with dynamically updated heuristic schedules. Moreover, there are other sources of production uncertainty, including new rush orders, canceled orders, machine breakdowns, rework, material shortages etc. They are not considered in the assumptions of most algorithmic scheduling models and they may cause further worsening in the quality of algorithmic solutions compared to simple dispatch heuristics.

Kutanoğlu and Sabuncuoğlu (2001) investigated the iterative simulation-based scheduling mechanisms with similar perspective of Lawrence and Sewell (1997). The effects of stochastic events, such as machine breakdowns and processing time variations, on the scheduling system performance are examined. It is stated that simulation based heuristics perform better than standard dispatching heuristics, and are more advantageous in the dynamic and stochastic environments. However, the effect of scheduling algorithm deteriorates with the introduction of uncertainty. As a result, in the dynamic

and stochastic environment, dynamically updated heuristic schedules perform better than static optimal sequences. In particular, dynamic and state dependent corrections may provide a simple dispatching performance algorithm make well.

Sabuncuoglu and Kizilisik (2003) worked on a centralized solution methodology for flexible manufacturing systems. They developed a simulation based scheduling system to study reactive scheduling problems in dynamic and stochastic manufacturing environments. They classified the scheduling decisions as when-to-schedule and how-to-schedule. When-to-schedule determines the timing between two consecutive scheduling points, and how-to schedule determines the way of generating feasible schedule. Their simulation results indicate that the variable time response is better than the fixed time-response, and the full scheduling scheme generally performs better than the partial scheduling. They also showed that on line scheduling is more robust to uncertainty and variations in processing times than the optimum seeking off line scheduling

Kutanoğlu and Wu (2004) studied methods to improve scheduling robustness under processing time variation for classical job shop problems. A combination of static and dynamic scheduling is proposed as a two stage method. The first stage is called as the preprocessing stage where they identify the critical decisions to global performance under the presence of random changes and disturbances. These decisions are optimized by using a priori stochastic information and Lagrangian relaxation. The next stage is the dynamic adaptation stage, where dynamic scheduling techniques are applied to overcome the effects of changes over time. Kutanoğlu and Wu (2004) showed that the developed PFSL (Preprocess-First-Schedule-Later) scheduling scheme is more robust than static optimization schemes while outperforming best known dynamic heuristics in both performance and robustness.

2.2. Negotiation Mechanisms for Distributed Systems

In dynamic problems the main issue is to determine "when to schedule/reschedule", however, in the face of decomposition the problem brings in additional complexity because the system is not purely separable. Inseparable constraints are introduced as

coupling constraints depending on the decomposition techniques. In general, there are distributed entities that must make decisions based on local information and on limited (and mostly invalid) information about overall problem. In order to resolve the conflicts of the system, coordination, cooperation, and communication mechanisms are proposed in literature. These negotiation mechanisms try to synchronize all the units in the system to achieve the same common goal i.e., globally consistent system.

Liu and Sycara (1997) present a multi-agent problem solving model and a coordination technique for job shop scheduling. In the model there are a group of agents; each agent is associated with either a job or a resource. They stated that s solution to a production scheduling problem is the result of coordinated conflict resolution in the iterative and asynchronous multi-agent decision making process. Liu and Sycara (1997) extended their study on the approach that there is at least an "order parameter" that separates problems into soluble regions for constraint optimization problems. Thus they define the "disparity composition ratio" which is the ratio of the number of bottleneck resources to the number of resources in the shop, to divide problems into subsets. Agents that are assigned to bottleneck resources are more constrained than agents that are assigned to non-bottleneck resources in the negotiation process. Coordination strategies are designed according to the most constraint agent. Their Coordinated Negotiation Agents (CONA) technique performs well compared to other constraint based heuristic search scheduling techniques. Moreover as a trail result, they showed that increasing coordination information decreased total number of cycles, i.e. with more information the system solves the conflicts more rapidly.

Gou et al. (1998) present a decomposition-coordination solution method based on Lagrangian relaxation. The method is proposed to provide a theoretical basis for guiding the cooperation among holons. The key idea of Lagrangian relaxation method is decomposition and coordination, where decomposition is based on the separable models, and coordination based on the pricing concept of the market economy. The hard coupling constraints are then relaxed by using Lagrangian multipliers or "shadow prices" in the economics literature. Since the original problem is separable, the relaxed problem can be decomposed into many smaller sub problems which are much easier

to solve as compared to the original problem (Gou et al., 1998). It is claimed that method leads to globally near-optimal performance. However, the communication protocols related asynchronous implementation issues, and the integration of physical and informational components of holons are not addressed.

Jeong and Leon (2005) approach to the subject from a different perspective. They worked on a special case of a single machine problem. In the problem the single machine is shared by multiple distributed sub-production systems which have the decision authorities and information. There must be a cooperation among these sub-production system to achieve a global goal of minimizing a linear function of the completion times of the jobs; e.g., total weighted completion times. However it is assumed that neither the sub-production systems nor the shared machine have complete information about the entire system. The formulation of the problem is given as zero-one integer programs. Their solution approach is based on Lagrangian relaxation techniques which are modified to require less global information. It is concluded that a global upper bound is not necessary, in other words there is no need for a single master problem that has a complete view of all the coupling constraints (Jeong and Leon, 2005). It is experimentally shown that the proposed methodology exhibits a promising performance compared to the Lagrangian relaxation with a subgradient method, besides it can be applied to situations with more restrictive information sharing.

There are also bidding algorithms for decentralized coordination. For example in the autonomous agents architecture presented in the Caridi and Sianesi (2000)'s paper, the agreement about which part has to be worked by which machine is made through a message passing among agents. There is a bidding protocol among the agents to achieve mutual agreement. Caridi and Sianesi (2000) claimed that by a distributed control, new degrees of freedom (for example, the possibility of working the same product on more than one assembly line indifferently) or new objectives (for example, the minimization of setup time) can be easily introduced without heavy modifications of the architecture.

Anussornnitisarn and Nof (2005), develops a similar negotiation model with a

bidding algorithm. In the model Agents associated with activities (task agents) request agents of resources for the resource they require. The resource agents propose their bids to a task agent, including price and schedule information for each particular task. The task agents and resource agents communicate by using a coordination protocol to exchange information and decision. The environment is not competitive but cooperative. Each resource agent will try to provide its service to task agents. Bid pricing algorithms of resource agents and bid selection algorithms of task agents are defined in the paper, and market based pricing and costing approach is applied in the decisions. There is no direct communication among the group of task agents and among the group of resource agents. The only direct communication channel is that each task agent can send messages to all of the resource agents and vice versa.

In Kutanoglu and David Wu (2006), instead of resource agents competing to win job requests, it is the case where job agents compete for scarce resources (machine-time slots). Agents may not share private information about their state, and it results the case of asymmetric information for the independent agents and the issue is incentive compatibility to align behavior of agents in favor of overall system efficiency. The negotiation mechanism is proposed as an extensive study in the game theory literature. Schedule generation and schedule valuation are separated steps. They design a 'schedule selection game' where all participating agents state their preferences via a valuation scheme, and the mechanism selects a final schedule based on the collective input. Kutanoglu and David Wu (2006) contributed to the existing literature by studying a more general case of a multi-stage, multi-machine environment with a set of pre-generated candidate schedules.

Wang et al. (2003) proposed a heterarchical multi agent system and distributed ruler-based scheduling mechanism. By the discrimination among agents and rulers, the scheduling problem is divided into two groups: decision-making problems for each individual agent, from the agent's own perspective, and coordinating problems between agents for the global goals of the whole system. Rulers are encapsulated in agents; therefore, the complexity of construction is reduced and rulers can be reconfigured and reused easily with agents regrouped dynamically. The simulation results have

successfully shown that proposed approach is capable of generating feasible schedulers in agile manufacturing environments.

Finally for this section one of the most recent literature review papers about the applications of agent-based systems is investigated. Shen and Norrie (2006) focused on agent based systems for intelligent manufacturing, namely on enterprise integration, enterprise collaboration, manufacturing process planning and scheduling, manufacturing shop floor control, and holonic manufacturing systems. They also discussed some key issues in implementing agent-based manufacturing systems such as agent encapsulation, agent organization, agent coordination and negotiation, system dynamics, learning, optimization, security and privacy, tools and standards. They called attention that there are very similar studies to those completed before and there is no significant improvement since 1998 for some difficult problems such as full integration of manufacturing process planning, scheduling, and control, particularly integration with real time information from data collection systems. Shen and Norrie (2006) suggested to focus on the integration of agent-based planning and scheduling systems with existing systems used in manufacturing enterprises.

2.3. Distributed Modeling Techniques

In this section, consistency problem is investigated apart from negotiation mechanisms, instead the aim is to provide the central attention to shared data, each of which is related with and shared various agents. The aim is to investigate inspiring approaches that will be background for the our proposed model.

Campos and Navarro (2004) present a causally consistent Distributed Shared Memory (DSM) protocol with main feature as less communication overhead. Distributed Shared Memory (DSM) provides a virtual address space that is shared among processors in a distributed system. Instead of providing locking mechanism shared memories are replicated in order to increase efficiency. Since fewer restrictions are imposed to the replicas, more efficient implementations are possible, but application programming becomes more difficult. Causal consistency is a model that offers a bal-

ance, by allowing efficient implementations without significantly increasing programming difficulty (Campos and Navarro, 2004).

Mahoney et al. (2004) studied on the modeling of reactive objects using state charts. He searched for a methodology and framework that allows independent state charts to be composed into a single model. The ideas are emanated from Virtual Finite State Machine (VFSM) implementations. VFSM separates platform independent behavior from platform specific behavior. Platform specific behavior is modeled in the form of input and output processor classes that interact with the VFSM. But there is a high degree of coupling between the state machine and the processors. Mahoney et al. (2004) realized that platform specific details can be modeled as crosscutting concerns by using Aspect-Oriented Modeling techniques, thus implementations that have less coupling and more reuse possibilities than traditional VFSM implementations, are achieved. Thus Mahoney et al. (2004) propose a state chart with orthogonal regions which is composed of two or more independent state charts that run concurrently and broadcast their events. However he recognized as insufficient part of his study that the representation of the model he proposed is involute.

Most distributed systems use the communications layer to record and log particular kinds of messages (also called events) that indicate the performance of basic operations. Events that indicate related activities in the communication layer may appear to be widely separated by other events and by the time. And there may be a lot of irrelevant events mixed in with ones of interest. It is hard to analyze and interpret the event logs at the communication level (Luckham and Frasca, 1998). There is a need to relate sets of communication events with higher level operations that the objects perform in order to provide information about what component objects of the system are trying to do at the application level.

Luckham and Frasca (1998) stated the goal as: Filtering interesting events, Aggregating sets of low level events into corresponding abstract higher level events that they signify, Detecting causal relationships between events that happened at different times in various subsystems, and Monitoring event logs and automatically detecting

violations of critical requirements.

Events are generated by the system and they are like messages in form (they contain data etc.) but in addition they also denote activities (such as a change of state of a component or the start or completion of a task). Abstraction Hierarchy is to separate the system's activities and operations into layers. The maps between sets of events at different levels must be defined in order to specify a hierarchy completely. Event pattern languages are a fundamental technology for extracting information from distributed message based systems. They are basis for specifying abstraction hierarchies by means of event aggregation maps, and for automated monitoring and aggregation of data from communication layers. In the paper, The Rapide event pattern language (they named) includes causal and timing relationships between events (Luckham and Frasca, 1998).

3. PROBLEM DEFINITION

The major problem in distributed models is to resolve the system conflicts in an effective way. As mentioned in the introduction chapter for distributed systems there are various traps that can lead to inconsistencies. In this chapter, reasons for system conflicts are investigated and they are formulated in mathematical terms.

The methodology behind the study is object orientation. In object oriented systems, an object is an organized entity that has attributes, behaviors. An attribute refers the data encapsulated within the object, whereas behavior refers the method used in the system functions. The state of an object is an instance of the object attributes, i.e. the values assigned to the attributes in an instance defines the state of the object.

Definition:

The state of an object "o" at time "t" is denoted by $S_t(o)$.

An object may live in some states, these are "stable states". Stable states define the current situation of the object. While the object is in stable state it may perform its state-defined actions, and these actions do not cause any change in the definition of its current situation. For example, when a machine processes an operation, it is in "in-use" state. During the time that machine is in in-use state, it performs one of the actions which are specific to in-use state, such as processing a task, and the state of machine is unchanged.

However there are some instances where the object is in a transient and undefined situation, these states are called "unstable states". While the object is in unstable state, it may also perform some actions such as actions performed during the state transitions. As an extension of example above, when machine completes an operation, it makes transition from in-use state to idle state. During this state transition machine

is neither in in-use state or in idle state, thus this machine is said to be unstable (if the situation is not required to be defined for example as "setup state").

There is not a strict distinction between the definitions of stable and unstable states. Stability is not related with the duration of time that object spends in a situation but appears as a consequence of the necessity to define the situation which depends on the case objective.

The object is in a stable state when momentary data encapsulate within the object has a meaning and the values of attributes defines the current situation of the object in the face of a specific scenario. The application of some actions on the object, may simultaneously or consequently change the current state of the object, so the object is said to be "unstable" during such applications. In other words, there are events result the application of some transformations Ω on the object o. These transformations may instantaneously change some attributes among the ones which defines the state of the object. Events may also trigger a transition from state to state and the object performs certain actions as transformations during these transitions (such as assignment of variables). In any case the object is unstable during the application of transformations.

Let $APPLY_t^{\Delta}(\Omega)$ indicate initiation of transformation Ω at time t, which is expected to take Δ time units.

Definition:

$APPLY_t^{\Delta}(\Omega)$ to object o. If $\exists T$ such that $S_t(o) \neq S_T(o)$ for $t \leq T \leq (t + \Delta)$, then object o is said to be ***unstable*** in $[t, (t + \Delta)]$.

Problem 1:

Let the object o be unstable in $[t, (t+\Delta)]$ and $APPLY_x^{\Phi}(\Theta)$ to o. Problem occurs when $[t, (t + \Delta)] \cap [x, (x + \Phi)] = \emptyset$.

The first problem given above is related to simultaneous access of multiple process on the same data encapsulated within the object. In order to provide consistent representation, coordination of concurrent processes should be provided.

The representation of data within an object is sometimes required to be updated. The updating process is simply reading the data from another domain, for instance from the data base object p, and transforming it to the local domain of object o. The necessary transformation to apply updating is called "synchronize" and is denoted by $sync(o, p)$. Indeed the aim is to imply the same situation in dissimilar domains. Thus not one to one equality but the equivalence between the current states of the objects is required. For example the object p has a capacity attribute which can be assigned a value between 0 and 10; the object o in another domain also has the capacity attribute but it is assigned as 0,1,2 which mean insufficient, sufficient and excess respectively. When capacity is equal to "10" for the object p, it is assigned as "2" for the o, but the reverse is not necessarily true. Moreover equivalence may be related with combinations of various attributes. The equivalence operator is defined as a one-sided function.

Definition:

Let the sign "$\mapsto$" denote the equivalence operator. If $S_t(o) \mapsto S_t(p)$ then the state of object o at time t represents the state of object p at time t.

Definition:

Let transformation Ω be $sync(o, p)$. $APPLY_t^{\Delta}(sync(o, p))$. If $S_{t+\Delta}(o) \mapsto S_{t+\Delta}(p)$ then it is ***perfect synchronization***, means that the state of object o at time $t + \Delta$ represents the state of object p at time $t + \Delta$.

Problem 2:

There is no perfect synchronization since system is not perfectly observable.

The second problem occurs because the representation process (i.e synchronize transformation) takes time and when it is finished, that instance can be changed and thus, does not exist with the initially given conditions anymore. As mentioned in the introduction, the reason of the change may be due to dynamic nature of the environment likewise existence of other processes related to the same data. In any case, actual situation may change but necessary transformation to signify the change is blocked during the synchronization. Blocked transformations are not perfectly observable in distributed systems. In a distributed environment making observation is analogous to watching a tennis game; the observers elude many ongoing events while the focus is on the ball.

For example, a scheduling object requires various data, which most probably is stored in a corporate data-store (ERP, RDBMS, etc.), regarding a job in order to plan its operations. Suppose that loading data takes Δ time units. If scheduling object starts reading the data regarding the job at time t, and completes the process at time $t + \Delta$, the question is: how accurately does $S_{t+\Delta}(o)$ represent the state of respective job at time $t + \Delta$?

Moreover two agents o and p may have loaded the data regarding the same job (probably at different time frames). Hence, $S_t(o)$ and $S_t(p)$ have some representation of the same job. How similar or dis-similar are these representations?

Definition:

Let $S_t^{o_1}(p)$ denote the set of states of object p for the representation of the state of object o_1, i.e., $S_t^{o_1}(p) = \{S_t(p) : S_t(o_1) \mapsto S_t(p)\}$. Similarly for object o_2, $S_t^{o_2}(p) = \{S_t(p) : S_t(o_2) \mapsto S_t(p)\}$. If $S_t^{o_1}(p) \cap S_t^{o_2}(p) = \emptyset$ then there exists ***conflict***.

The definition above is related to the system conflicts due to different solution policies among the objects which try to solve the same instance of the problem. Since scheduling problem is not totally separable, there are bound to be some coupling constraints which make independent decisions made by objects to relate to each other.

Perfect synchronization is almost impossible to achieve but even it were not the case, i.e. even objects have the same representation of the problem instance, there may be still system conflicts because of the individual and different behaviors of the objects. That is, for two agents o_1 and o_2, $S_t(o_1)$ and $S_t(o_2)$ may include decisions which are related to each other through some coupling constraints. The question is, how compatible are these decisions?

Let the transformation Ω be conflict resolution function, *Conf.Res(o_1,o_2,p)*. In order to resolve the conflict $APPLY_t^{\Delta}(Conf.Res(o_1, o_2, p))$ should be defined.

Definition:

Suppose $S_t(o) = S_{t+\Delta}(o)$ but $S_t(p) \neq S_{t+\Delta}(p)$. If $S_{t+\Delta}(o) \mapsto S_{t+\Delta}(p)$, then representation of object p for object o is ***robust representation***.

The last definition is the goal to be achieved. All possible states of an object should be defined in such a way that the representation of the state remains consistent notwithstanding to the system changes. In other words representation should be robust to system variances.

4. MODEL

In object oriented systems, each object is capable of receiving messages, processing data, and sending messages to other objects. Each object can be viewed as an independent little machine with a distinct role or responsibility, and collaborates to other objects to achieve the goal within the framework of a common task.

Object orientation supports separation and encapsulation of concerns into distinct objects that overlap in functionality. For example, procedures, packages, classes, and methods all are defined to encapsulate concerns into single objects, thereby the activities of an object are determined.

The activities of an object can be further decomposed into different aspects. This approach is an extension of a method known as Aspect Oriented Software Development in literature. AOSD address that some concerns are not specific to a single entity but they cut across many modules in a program. These concerns that deny encapsulation are called crosscutting concerns. AOSD aims to gather these crosscutting concerns in separate modules known as aspects by providing means for systematic identification, separation, representation and composition. Thus localization can be supported, and results in better support for modularization hence reducing development, maintenance and evolution cost (Rashid et al., 2006).

In the proposed model, one of the aims is to observe the activities of objects in an effective way at any instance. From the modeling perspective, a state chart is an effective design tool to investigate reactive objects in dynamic scenarios. state chart indicates the stable states in which an object may live, the events that results state changes and the actions that occur during these state changes, so it provides an effective monitoring tool to observe the actions of the object.

The activities of an object can be separated into distinct aspects according specific to concerns. This approach is an extension of AOSD. Each aspect of an object is

modeled in a different state chart where each state chart is a representation of one possible parallel universe (parallel/ orthogonal state charts, crosscutting concerns). The advantage is that there is now a clean mapping of instances of activities in favor of more effective monitoring and simple control mechanism because, independent activities are prevented to be mixed in a single state chart as a cross product, instead two or more independent state charts can run concurrently where each state chart describes the dynamic behavior of the object with respect to its related aspect.

Effective monitoring supports easily adjustment of certain object parameters, besides the control of object functionalities in a common task in order to analyze various scheduling solution approaches in case of different scenarios.

4.1. State Base Modeling

In object oriented systems, an object has attributes, behaviors. An attribute refers the data encapsulated within the object, behavior refers the method used in the system functions. The state of an object is an instance of the object attributes, i.e. the values assigned to the parameters of attributes in an instance defines the state of the object.

The activities of the object are separated into distinct state charts corresponding to different aspects, so each object state chart is the conceptual definition of the aspect in a sense. In terms of a particular activity about an aspect of the object, there are stable states in which an object may live, there are events that trigger a transition from state to state and object performs certain actions during these transitions. In other words, the object is in a state when momentary data encapsulate within the object is in some combinations of attributes, has a meaning and defines the current situation of the object.

In each state chart, the object can be in only one state. For example, there are three possible states for the machine object to signify the current situation in the shop floor, machine can be in one of the “idle”, “in-use”, and “broken” states.

Some events may trigger a state transition. The decision on what action to take depends on the event and the current state of the object. Considering current state of the object, same events may cause different actions. For example, when machine is in idle state, "the operation start" event trigger a state transition to in-use state and necessary actions to provide the transition are realized; but when the machine is in broken state the same operation start event result no actions, until "repair completion" event occurs. Obviously different events result different actions for the same current state of the object. For example when machine is in-use state, the "breakdown" event trigger state transition to "broken" state.

There may also exist events that trigger not a state transition but an application of some other actions in the current state of the object. As an extension of the example above, when the machine is in in-use state, if the "operation arrival" event occurs, the object performs necessary action i.e. taking the operation into queue while it is still in in-use state.

All possible transformations of an object are defined as the action of the object. The actions of objects are defined in such a way that object performs one possible process at a time, thus multiple processes which can be triggered simultaneously are excluded. In other words, the action of an object identify the process which is mutually executed and the actions that can be concurrently performed are processed in distinct state charts.

As an example, consider again a machine object in in-use state. If the "operation stop" event occurs it will stop the current operation and make transition to idle state; and if the "breakdown" event occurs it make transition to broken state. These operation stop and breakdown events can not be processed at the same time. The event which occurs first is considered and other is not allowed to be performed. However "operation arrival" event can be processed concurrently because taking the operation into queue is a distinct action, thus defined in another state chart so both operation arrival event and one of the breakdown or operation stop event can be handled at the same time. As another example, situation is similar to pay money into an account or

to draw out of an account; these processes can not performed simultaneously, anterior one is processed first; however application for an new account can be processed concurrently with pay into an existing account or drawing on an account.

In the proposed model, a state chart operate as an algorithm, thereby provides a control tool, because the current situation of the objects are now easily observable. If an object is in one of the states, the object is ready to perform one of the actions followed by the possible events for the current state of the object. However during the execution of the expression in an action, the object is in unstable state in respective state chart and is unavailable to perform another process.

From the programming perspective, considering the synchronization issues, locking mechanisms can be introduced to relevant state charts by identifying the corresponding object as being stable or unstable. Moreover the access to the shared data within an object is limited by separating the aspect of objects in distinct state charts. In other words, the proposed state chart base modeling provides the focus about the control on efficient, small and distributed shared data, each of which is related with and shared by a limited number of objects whose activities are easily observable.

4.2. Processing of System Messages

In distributed systems, objects need to coordinate their activities with other objects in the system by sending and receiving messages. The communication is required to integrate and sequence the partial processes among distributed entities, besides the communication need appeared as a result of coordinated conflict resolution in asynchronous multi-agent decision making process for non-decomposable production problems.

A system message includes a breakdown of a machine, repair completion of a machine, due-date change of an order, new order and so on. Each system message is related to one object or a group of objects that are associated in functionality for a common task.

In the proposed model, there is a distinction between the objects that perform an activity and the objects that organize communication. Objects which administer the messaging mechanism are called **agents**. Agents are capable of receiving the relevant messages, processing messages by organizing the relevant objects that perform a complementary activity for a common task, and broadcasting the resulting messages.

Agents keep objects distinct from messaging issues and objects do not communicate without an agent connection. Each agent may be related to a class of objects or a group of classes of objects. For example shop floor agents manage machine and operation objects in the shop floor environment which is a package, whereas scheduling agents manage objects such as local schedulers in scheduling environment which is also a package. Agents listen all relevant messages to objects which are assigned to itself. Interesting messages are classified by their context, and each message is processed respectively.

The activities of objects are defined in their state charts corresponding to their aspect. In response to a system message, agent operates relevant state charts in a logical sequences. The state charts are connected based on system messages into one working engine which is called **Message to Event Processor** (MEP). Each MEP is related with a specific message. For example the MEP that respond to breakdown message of a machine, is different from the MEP that respond to repair completion message of the same machine. Thus one agent may be composed of many MEPs, where each MEP processes the message in a specific context.

MEP is a handler of system messages, it interprets a system message as respective events for relevant state charts and triggers state charts by these events. This is event processing. An event is an interpretation of a received message in the context of a state chart. For instance, “breakdown of a machine” as a system message is related to the broken machine itself and also the current operation on it. The message is interpreted as “breakdown” event for the machine and as “operation stop” event for the operation which is currently being processed on the broken machine. These events in corresponding state charts trigger the relevant object to act depending on the current

state of the object.

MEP is also defining the logical sequences of activation of various state charts of various objects. For instance, again suppose that a machine is broken while it is processing a task, then two state charts will be activated, i.e. the state chart of the machine and the state chart of the current operation on that machine. The process can be designed such that firstly the machine changes its state and corresponding event is evaluated, then the events related to the current operation on it are handled. Thus MEP manages operation of the state charts by providing a logical sequence of activations.

By using the proposed mechanism, messages are allowed to be reinterpreted in different aspects as corresponding events, thus the dependencies between state charts are reduced. Besides MEP can provide event ordering by determining the sequence of the activities of objects it manages, in response to messages received.

4.3. Messaging Mechanism

In distributed systems, groups of similar or complimentary agents act together to solve problems based on their value systems and communication abilities. Agents organize objects that own their individual decision making tools and computing facilities specific to their local domains. As an example from the distributed scheduling perspective, scheduling problem is decomposed into sub-problems; then the functionality of scheduling decisions is shared by the local decision makers, i.e. scheduling agents. Scheduling agents may communicate through a negotiation mechanism to resolve conflicts due to asynchronous multi-agent decision making process for non-decomposable production problems. In this distributed scheduling environment, each job, machine, machine group, operation, order object and so on, can be assigned to an agent. Thus messaging mechanism should be provided to work properly irrelevant to the functioning of an agent.

Referring to Figure 4.1 an agent is designed to handle communication issues

apart from decision making problems. Each agent has a receiver to listen to messages and a sender to dispatch resulting messages. When a message is received, it is provided to the respective MEP corresponding to the context of the message and MEP processes the message. MEPs are encapsulated in agents and have connections to the related object which are assigned to the agent. By the proposed model, the complexity of construction is reduced because computing facilities of objects can be reconfigured easily.

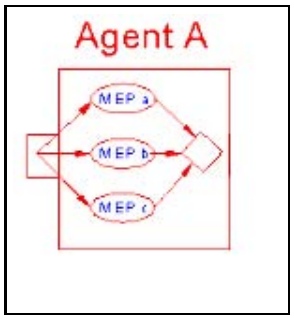

Figure 4.1. The representation of an agent

In the proposed model, another aim is to provide an operative and flexible messaging procedure independent of manners for processing the messages. The object activities are organized by MEPs that belong to an agent. Agents are related with a group of objects and handle the coordination of the assigned objects with other objects in the system by communicating with respondent agent. However, in the face of the assumption that distributed entities may exist in different domains, agents should be able to act like interface points of the software. The aim is achieved if some portion of agents are designed to act as proxy agents by providing an addressing mechanism.

The Figure 4.1 is given to show the functioning of proxy agents. If the objects which are assigned based on a concern to an agent, locally exist in the current domain, then their respective agent exist in this domain. However if an agent has no object connection in the current domain, then it holds the address of the actual agent, i.e. a remote address to find the respondent agent. For example referring to the Figure 4.2 when Agent A sends a message to Agent B, Agent B receives the message, but when Agent A calls for an agent with name Agent C, since Agent C does not locally exist in Computer X, Proxy Agent C receive the message and it directs the message to Agent

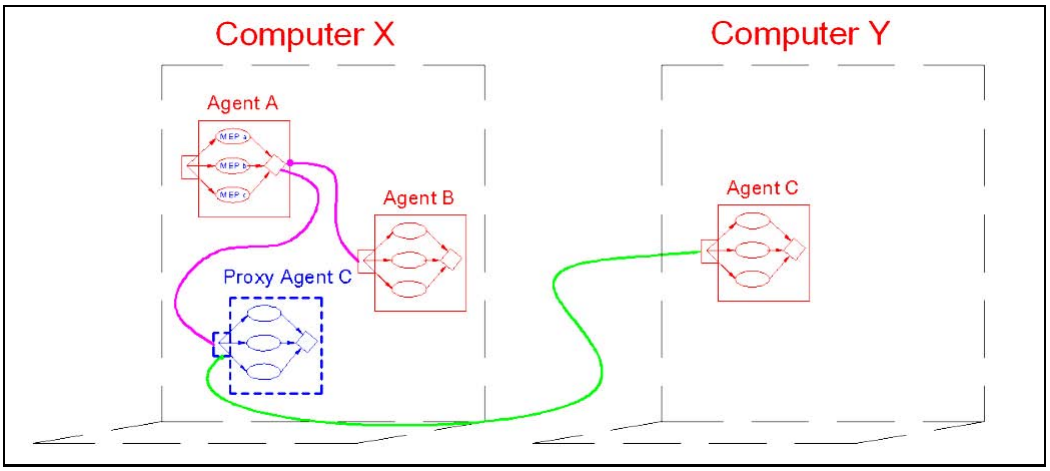

Figure 4.2. The messaging flow among agents

C which exists in Computer Y through a communication media.

Each agent has a type and a name. Type of the agent acts like a namespace. Each agent registers to the messaging channels with its type and code and holds a local address. The inter-domain communications are provided through this messaging channels. Proxy agents, however, hold also a remote address. In other words, if the agent respondent to a message exists in the domain where the message is sent, then the address of that agent is local, otherwise, it is a remote address and respondent agent is proxy object for the current domain.

4.3.1. Receiving Messages

Message sender agent is not interested in "where the receiver agent is", and "what the listening protocol of receiver agent is". Sender calls the receiver agent by defining the type and the name of the receiver agent. The proxy agent works like an interface and it directs the message to the remote address it holds, so the communication is provided through the remote address that proxy object holds. The address and protocol that is used to transform messages can be any standard messaging protocol like message queues, TCP/IP, etc.

Received messages are handled by the agent. Agents register to the channels with their names and each agent distributes the received message to corresponding MEP. In

other words, MEPs are classified according to the context of the messages that agent processes, responsible MEP processes the given message.

4.3.2. Sending Messages

As mentioned previously, each agent is related with an object or a group of objects. Agent knows which messages to respond. After processing messages by corresponding MEPs, the resulting messages are broadcast to the respondent agent for the continuity of the process. Each agent has an algorithm called "Dispatch Messages". This algorithm encapsulates all possible messages that can be sent into a single description. It also give the answer of the question that which agent receives the resulting messages.

Messages are created by MEPs of agents. Agent dispatches messages. MEP specifies the message code and the condition that the message should be created. Agent specifies the name and type of the receiver agent, the content of the message and the condition that the message should be dispatch. In other words, the composition of messages and the distribution of messages are two separated process. Messages are created in certain conditions without considering respondents of the messages, and in dispatch message algorithm the respondents are determined, the message content may reformed, and the message is distributed.

5. ICRON IMPLEMENTATION

5.1. Preliminaries

In this study, ICRON software is the framework to evaluate and implement the proposed model. ICRON is a general purpose visual algorithm modeling tool. It is used to construct optimized generic algorithms. It is based on object orientation and includes nodes and links that are developed in C++ and XML. It has a very special modeling system called GSAMS.

GSAMS is an object-oriented graphical scheduling algorithm modeling system which is used to construct and modify scheduling algorithms within graphical scheduling environments. GSAMS allows users to drag and drop algorithmic units on the design window, and visually construct algorithms, thereby ICRON provides the user to construct flowchart-like models without knowing any special computer language.

In order to model an algorithm, the user works on nodes. Each node represents a particular operation on the current object. Nodes have input and output link points. Objects flow between the nodes through link points and all of the data manipulations are managed by node and link structures of the algorithm model.

In the study, Advanced Planning and Scheduling (APS) module of ICRON is used. In APS the system objects are classified into a set of distinct system components. Each component models a particular functionality of the scheduling system, and uses a well-defined communication mechanism to co-operative with other components. These system components are Psychical System Component, Product System Component, Job System Component and Scheduling System Component.

The user can benefit from existing system component definitions of APS, and also can define new components or extend existing ones.

5.2. State Chart as an Algorithm

In the proposed model, each object may have more than one state chart. A state chart defined for an object represents the possible states of the object, events that this object may need to respond in the form of actions, and possible state changes as a result of these actions. Each object state chart is the conceptual definition of the aspect in a sense, and introduced to the model as an algorithm. The state chart is a composition of three specific types of nodes:

- **State Node** : It has a name, i.e. state code and it has a conditional expression to check if an object is in this state or not. State nodes for an object in a state chart must be defined such that the conditional expression of only one state node returns true value at a time.
- **Event Node** : State nodes are followed by event nodes. Event nodes are just the name of the events. They connect the action nodes and state nodes. Event nodes act like conditional branching since for a particular state, different events may cause different actions, in addition to that, event evaluated in one state of the object may cause a different action than the same event in another state.
- **Action Node** : The decision on what action to take depends on the event and the current state of the object. Action Nodes are defined to indicate the actions in the case of the event triggers a transition from current state. Action nodes are designed to execute an expression in its field.

A state chart represents the current situation of the linked object at an instance. The condition expression in a state node is a function of an attribute or a combination of attributes of the current object. If the values assigned to the respective attributes for a specific state has meaning, then these attributes defines the current state of the object. Object may be in one of the states and available to perform an activity in case of a specific event received i.e. event is related to its current state or may already perform an activity which may simultaneously or consequently change the current state of the object. While the object is executing a particular operation in the action node it may be unstable state, thus unavailable to perform another process.

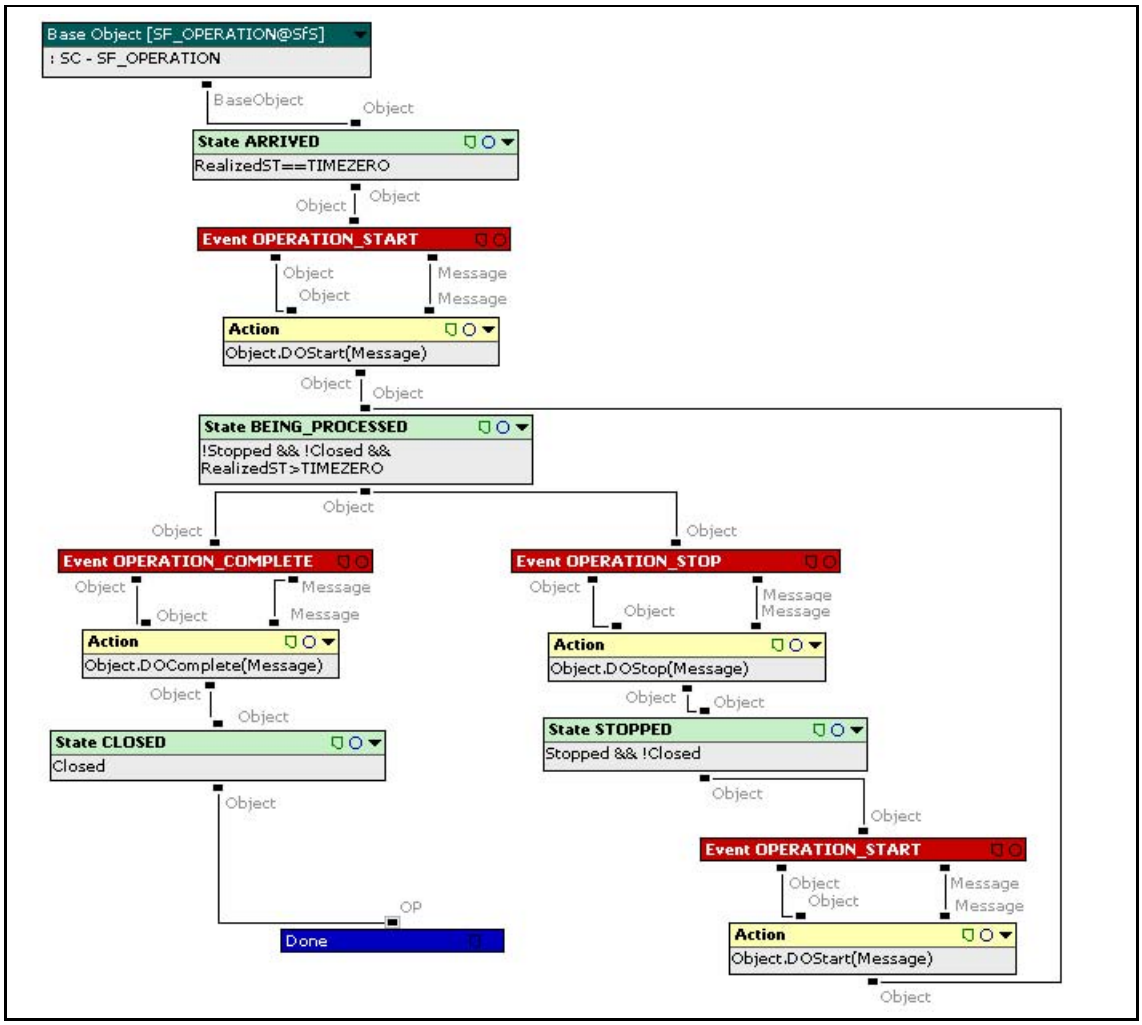

Figure 5.1. The state chart of an operation object in shop floor

In Figure 5.1 a declarative model of a state chart in ICRON is presented. The state chart belongs to an operation object in shop floor model. As seen from the figure, an operation object can be in one of "arrived", "being processed", "stopped", and "closed" states. When the operation object arrived at one of the machines in shop floor , the attribute to define the realized start time of processing the operation (RealizedST) is set as the default value TIMEZERO, so it is in arrived state initially. There are also attributes called stopped and closed to denote if the processing of mentioned operation is stopped or if the mentioned operation closed respectively, and initially these attributes are set to return false value. In each state the object waits for consequent event among "operation start", "operation stop", and "operation complete" events, then with the realization of the expected event, the object starts to execute the action defined in the action node.

The operation makes transition from its initial arrived state to being processed state when operation start event is realized, so in the following action node the RealizedST value is set to the corresponding event time, so the condition expression of being processed state returns true value after the action node completes the execution of its own transformations. The object may receive the operation stop event in case of a breakdown of the current machine or may be it is decided to start another operation so move the present operation and so on. After the operation is completed, the closed attribute is set to return true value, and the closed state is the end state for an operation object.

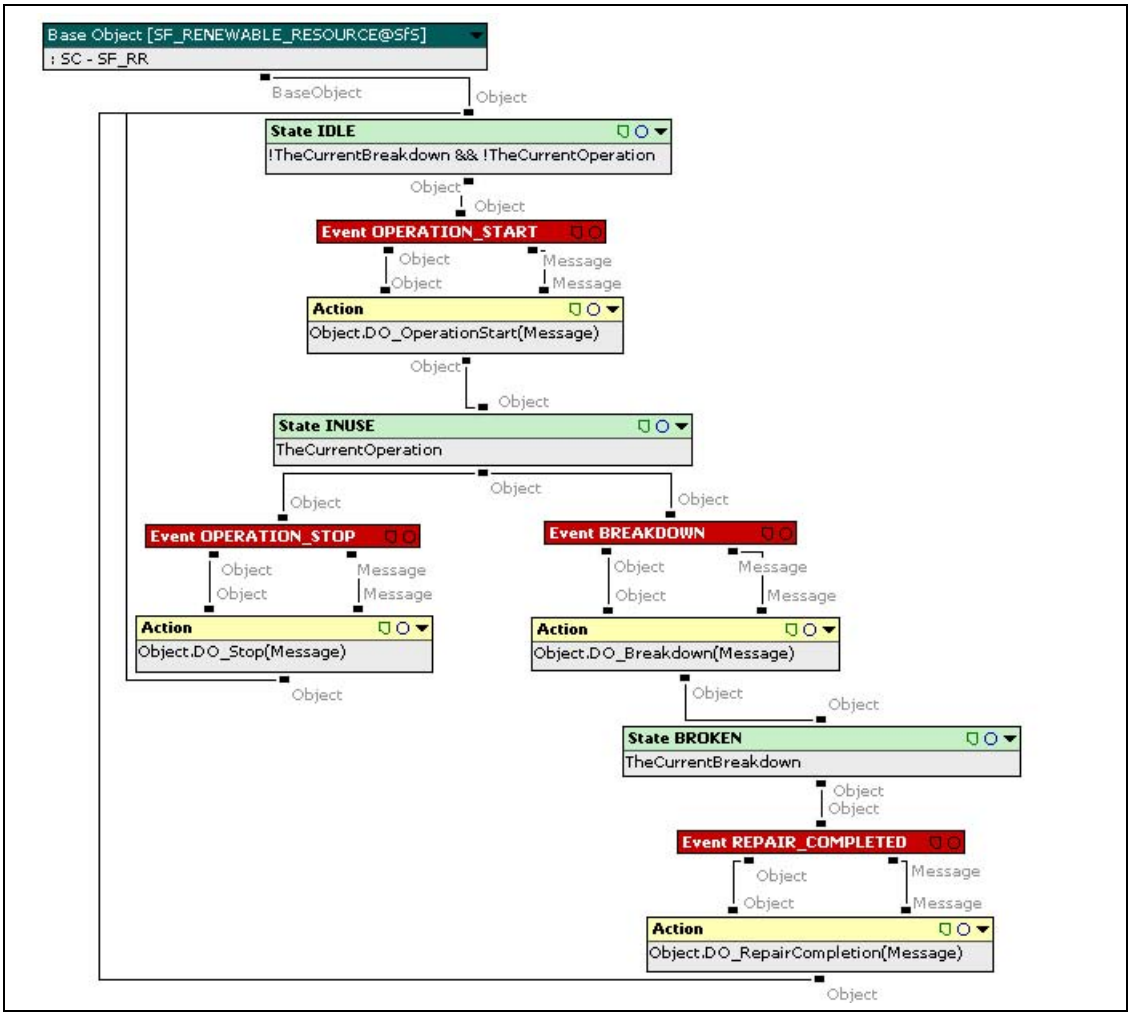

Figure 5.2. The state chart of a machine object in shop floor

The Figure 5.2 is another model of a state chart which is related with core concerns of a machine object. As seen from the figure the machine object may live in one of the "idle", "inuse" and "broken"states. There are two attributes to denote the

current state of the machine object: "TheCurrentBreakdown" and "TheCurrentOperation". If the machine is not broken and it is not processing an operation, then these attributes are null and the machine is in idle state, if there is an operation which is currently being processed on the machine, the TheCurrentOperation attribute is assigned to "1", it is similar for breakdown state. When the machine is in idle state, it is ready to start an operation and it waits an operation start event to occur. However when the machine is in broken state the same operation start event result no actions, until "repair completion" event occurs.

All possible responses to events are defined in the form of actions of the object in such a way that object performs one possible process at a time in each state chart, thus multiple processes which can be triggered simultaneously are excluded. For example, in Figure 5.2 consider the machine is in inuse state. If the operation stop event occurs it will stop the current operation and make transition to idle state; and if the breakdown event occurs it make transition to broken state. These operation stop and breakdown events can not be processed at the same time. The event which occurs first is considered and other is not allowed to be performed. Thus, there should be an event ordering based on a logic such as processing the anterior event.

5.2.1. Parallel State Charts

As mentioned previously, each object may have many aspects related to the scheduling, data and communication. The activities of an object are separated into possible distinct aspects and defined in respective state charts which can be run concurrently. The action of an object identify the process which is mutually executed in respective state chart and the actions that can be concurrently performed are processed in distinct state charts. Parallel state charts may be specific to an object or may be not specific to a single object but they intersect many modules in a system. Data store manipulations are example for later class of parallel state charts.

The Figure 5.3 is a declarative example of a state chart for data store manipulations of objects in shop floor. The present object is a machine object but it is a

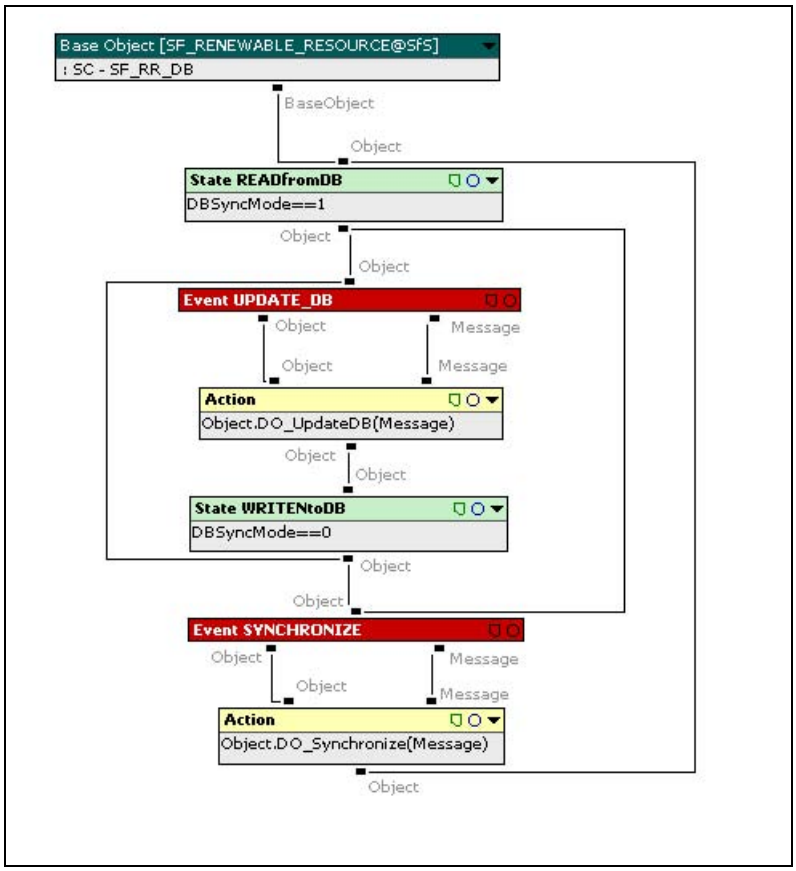

Figure 5.3. The state chart of an object in shop floor to declare data aspect

similar state chart (except the base object) for handling of data store operations of an operation object. After the object respond to an event by processing the consequent action in one of its other state charts, the data about the instance of the object which is encapsulated within the object may be changed. Thus the up to date instance should be written to the data store by generating update data base (UPDATE-DB) event. An additional scenario is that there exists other processes related to the data of the present object, i.e. the data of the object is shared in data store, and the instance of the mentioned data may be transformed by those processes, so the object needs to synchronize its current state by loading the data or part of the data from data store. This activity is triggered by SYCHNRONIZE event.

The state chart in Figure 5.3 for data aspect of objects provides systematic identification, separation, representation of data base concerns in a place. Thus localization is supported, and results in reducing development and maintenance tasks. For example the same state chart is also provided for operation object, but the expressions in the

action nodes (i.e. DO-UpdateDB and DO-SychronizeDB) are defined in different ways for each object respectively, in other words polymorphism is used. The data stores or the way to manipulate the data may be changed in time, this kind of situations are handled by modifying the functions in the action nodes.

There may also exist events that trigger not a state transition but an application of some actions in the current state of the object. The object perform an activity in response to a specific event and during the operation (in action node) object may or may not change its present state depending on the definition of the state. In other words the condition to perform an action is firstly depends on the current state of the object,and if the object is in a stable state then it is ready for an event to perform consequent action. However, for some aspects object does not necessarily wait the end of its action in order to accept a similar event to handle. A typical example is presented in Figure 5.4.

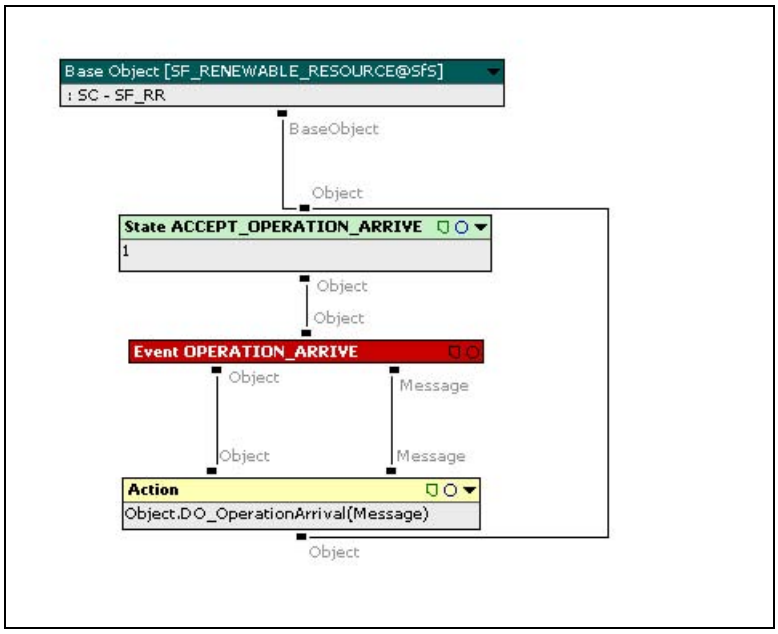

Figure 5.4. The state chart of a machine object for operation acceptance aspect in shop floor

Conceptually for the current aspect there is only one state and when the following event occurs, the respective action is performed. Since there is not a state transition the sequential assignments to respective attributes is not expected for most of the case, so the object is always stable, i.e. available to execute the action, in these kind of state

charts.

For example as in Figure 5.4 the condition expression of the state is always true (it is equal to one). If the object is asked what its current state is (for this state chart), then the object returns the state written in the state node even while it is processing the action. However if it were the case that the respective attributes to define the state is processed during the activity then the object will be in an unstable state (not in the state defined in the state node). So for this example the condition expression may be provided to some kind of functions related to transformations in the action node. These kind of separations are essential for mutual exclusion algorithms which are defined to prevent multiple processes on a shared data of an object.

Following the above discussions, the explanation of Figure 5.4 is as follows: "operation arrival" event can be processed concurrently because taking the operation into queue is a distinct action from the actions in other state charts, thus it can be handled at the same time with those actions. When the machine is in in-use state , (while it is currently processing an operation) in Figure 5.2, if the "operation arrival" event occurs in 5.4, the object can concurrently perform necessary action for operation arrival event i.e. taking the operation into queue. Since these processes are related with distinct part of the data within an object.

5.3. MEP as an Algorithm

Communication between agents are managed through messaging. "Messages" received by agents are interpreted by MEP (Message to Event Processor) definitions. MEPs of an agent are classified according to the context of the messages that agent processes, and each MEP is related with a specific message. MEP first decides which objects are related to the message. Then it maps the message to a respective "event" for the object, and triggers the state chart of the related aspect. Finally, MEP may create messages for other agents. In this framework, event is an interpretation of a received message in the context of a state chart, thus by using the proposed mechanism, messages are allowed to be reinterpreted in different aspects as corresponding events

and dependencies between state charts are reduced.

There are two nodes specific to a MEP: “Activate State Chart” and “Dispatch Message” node.

5.3.1. Activate State Chart Node

Each MEP is related to an object or a group of object. In each MEP the event codes for corresponding the state chart of the related aspects are defined, and the message is processed for a specific state chart with a specific event by Activate State Chart node.

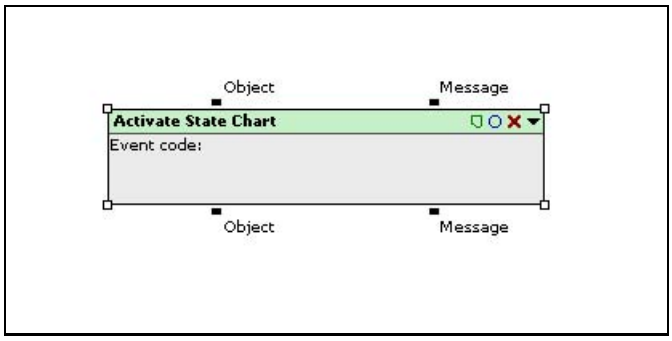

Figure 5.5. The Activate State Chart node

In Figure 5.5 an Activate State Chart node is shown. The Activate State Chart node is executed on an object with the input message text by defining two fields: state chart code and event code.

The steps of algorithm in this node is as follows:

1. All state nodes in the specified state chart are determined.
2. It is iteratively checked among the state nodes that if the condition expression of the current state node returns true. Stop when it is so, thus the current state of the object is found. (Note that object must be in one of the states in each state chart.)
3. The event nodes following the current state of the object are determined.
4. It is checked whether one of the following events to current state match the

occurring event (the event specified in the field of the node by MEP) or not. If it is so then the consequent action node is determined. (If it is not then the algorithm terminates without an action)

5. The expression written in the action node is executed by creating an instance of the algorithm.

After a state chart is activated with an event by activate state chart node algorithm, if there is an action to execute, then it is expected that the object will return its current state as the state that follows the activated action node. Thus, from the control side, verification of the state change is easily manipulated since when Activate state chart Node executes a state change, resulting state of the object can be checked by evaluation of the condition function of the state node.

5.3.2. Dispatch Message Node

After processing of a received message, other agents in the system can be informed by creating the resulting message and dispatching it to the relevant agents. This processes is handled by Dispatch Message node.

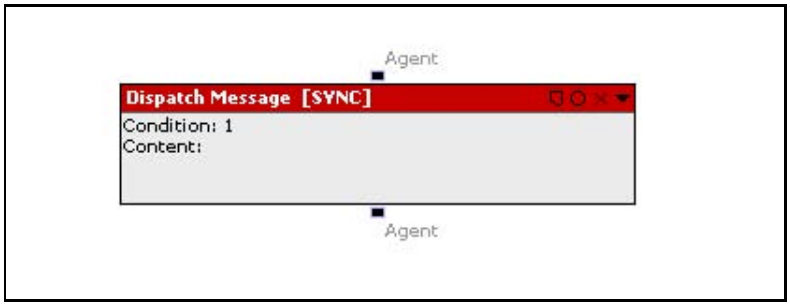

Figure 5.6. The Dispatch Message node

In Figure 5.6 a Dispatch Message node is shown. The Dispatch Message node is executed on an agent object by defining four fields: message code, condition, message content, synchronization mode. The fields of the node are used to specify the following:

- Message code: The message is dispatched through the name specified in this field.
- Condition: This field is needed to specify the condition that the message should be created

- Message Content: The expression written here returns a message text as string in XML form.
- Synchronization: It is provided to choose for options that either messaging is carried out asynchronously or messaging blocks sender until receiver is finished processing.

During the creation of a message, agent is not interested in "where the receiver agent is", and "what the listening protocol of receiver agent is", thus messages are created in certain conditions without considering respondent agents. The composition of messages and the distribution of messages are two separated process. In this node for this level, the agent is provided a message with a name and content to dispatch.

5.3.3. Processing of System Messages

Each MEP processes a message in a specific context. For example the MEP that respond to breakdown message of a machine, is different from the MEP that responds to repair completion message of the same machine. MEP first decides which objects are related to the message; from the implementation perspective MEP has pointers to the relevant object(s), thus MEP can reach the related state charts.

When message received from the environment arrives the MEP, messages are converted to events regarding the related aspect of object in a state chart and events are given to the related state chart of the object as an input in the form of event code. Thus an event is an interpretation of a received message in the context of a state chart.

The Figure 5.7 is a declarative example for a MEP definition. This MEP is specific to a breakdown message in shop floor. MEP points a renewable resource (machine) object as its local object. The current operation object is accessible from the machine object. As shown in Figure 5.7 there are four state charts to be activated regarding to the message : SF-OP state chart is for the operation object (previously explained in Figure 5.1) which is currently being processed on the machine, SF-OP-DB state chart is for the data store manipulations of operation object (it is similar

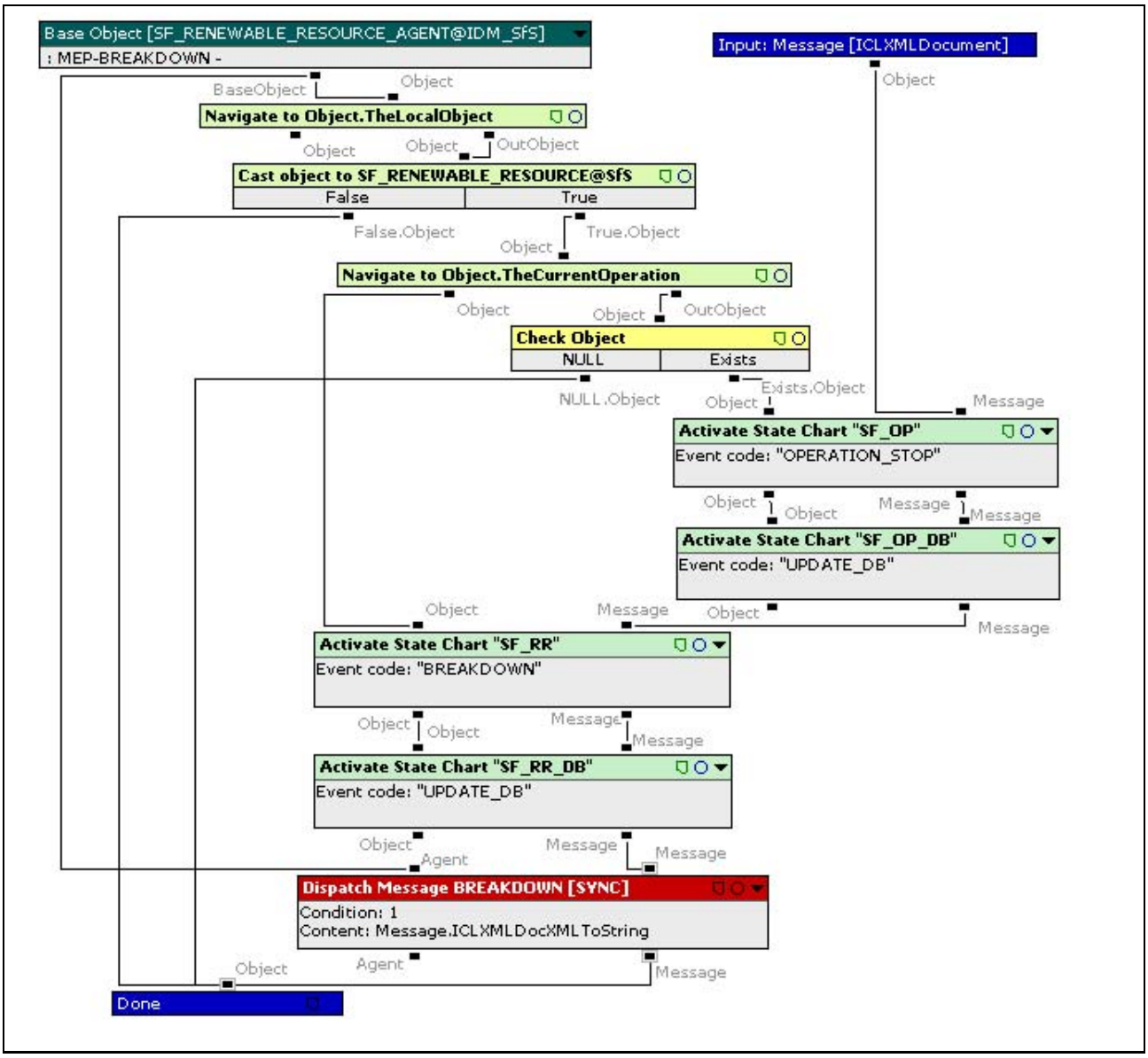

Figure 5.7. Message to Event Processor (MEP) for breakdown message in shop floor

to the state chart which is explained in Figure 5.3), SF-RR is for the machine object (previously explained in Figure 5.2) which is the currently broken machine, and finally SF-RR-DB is for the data store manipulations of machine object (as in Figure 5.3). These four state charts are connected based on the breakdown message into one MEP.

The message is interpreted as "BREAKDOWN" event for the machine, as "OPERATION STOP" event for the operation and as "UPDATE DB" (update data base) event for data manipulations of both operation and machine objects. Thus messages are allowed to be reinterpreted in different aspects as corresponding events and depen-

dencies between state charts are reduced. These events in corresponding state charts trigger the relevant object to act depending on the current state of the object by activate state chart node.

For example considering Figure 5.1 again, if operation stop event triggers a transition, then the action node which contains "DOStop(Message)" algorithm will be executed. In this algorithm necessary assignments are done sequentially and operation object makes transition to "STOPPED" state. Similarly considering Figure 5.2 again, if breakdown event triggers a transition, then the action node which contains "DO-Breakdown(Message)" algorithm will be executed, and at the end of the execution and machine object makes transition to "BROKEN" state. Moreover, considering Figure 5.3, update data base event triggers the action as "DO-UpdateDB(message)" algorithm which includes the necessary SQLs used for relational data base manipulations.

After the processing of breakdown messages for the objects in shop floor, MEP create messages for other agents as shown in Figure 5.7 by using Dispatch Message Node. Note that the content field of dispatch message node is the XML document that is received by MEP, in other words message is transported as it is received. However the content may be changed by calling a function which is written for this field specifically. To conclude, here for this level, it can be said that the message is composed and given to Dispatch Message Algorithm as an input, then agent dispatch the message to respondent agents. (Dispatching algorithm will be explained in detail below.)

It is worth to note that the logical sequences of activation of state charts are determined in MEP. For instance, the sequence in Figure 5.7 is that firstly the events related to the current operation on the machine are handled, then the machine changes its state by the evaluation of corresponding events. However it can be the case that both machine and operation processes are handled concurrently if the message is given to SF-RR state chart directly instead of waiting the message from operation object.

5.4. Messaging Structure of Agents

As mentioned previously, in the proposed model, messaging is handled by agents. Agents register to the message channels according to their types with their names. Type of the agent acts like a namespace (for example: shop floor agents). The channel specific to a type is branched according to the name of agents. Namely, there is an agent which listens to the channel at the end of each branch.

Message sender agent is not interested in "where the receiver agent is", and "what the listening protocol of receiver agent is". A message is send to the receiver agent by defining the type and the name of the receiver agent. When a message arrives to the channel, it triggers the Handle Message Algorithm of agents.

5.4.1. Handle Message Algorithm

Distributed entities may exist in different domains. Agents which do not exist in the current domain have proxy agents. The functionality of proxy agents are provided by an addressing mechanism. Each Agent holds either a local address or a remote address. If the receiving agent locally exists in the current domain then it directs the message to the local address it holds. If the receiving agent does not exist in the current domain then the receiver is the proxy of the respondent agent in this domain. The proxy agent works like an interface and it directs the message to the remote address it holds, so the communication is provided through the remote address that proxy object holds. The address and protocol that is used to transform messages can be any standard messaging protocol like message queues, TCP/IP, etc.

Before the explanation of the algorithm, it is worth to remind that MEPs are classified according to the context of the messages that agent processes, responsible MEP processes the given message. So each MEP has a context to identify the MEP.

Handle message algorithm is executed obviously on an agent object. The steps of the algorithm is as follows:

1. It is checked whether the address of the agent is a remote address. If it is so then the message is directed to the address and the algorithm terminates. If it is not then the agent locally exist in the current domain. The message is provided to the respective agent.
2. The message string is converted to the XML document, and then an object info (with the first line as message context).
3. It is filtered among the MEPs of the agent that if the message context match the context of the MEP (thus the responsible MEP is found).
4. An instance of the MEP algorithm is created and the message is given to the MEP as XML string.

In summary, agent distributes received messages to corresponding MEP by Handle Message algorithm.

5.4.2. Dispatch Message Algorithm

In order to send a message there are two steps: First the message is composed in MEP level, then it is dispatched by the agent to whom MEP belongs. The creation is explained previously in Dispatch Message Node section. MEP specifies the message code and the condition that the message should be created. Dispatch message node is connected to the Dispatch Message Algorithm of the agent. In this algorithm there is a specific node called Send to Agent Node.

5.4.2.1. Send to Agent Node:. Agent specifies the name and type of the receiver agent, the content of the message (the message content may be reformed) and the condition that the message should be dispatched by a specific node called 'Send to Agent" for the algorithm.

As shown in Figure 5.8 Send to Agent Node consists of four fields:

1. Agent Type: It is the namespace of the receiver agent.

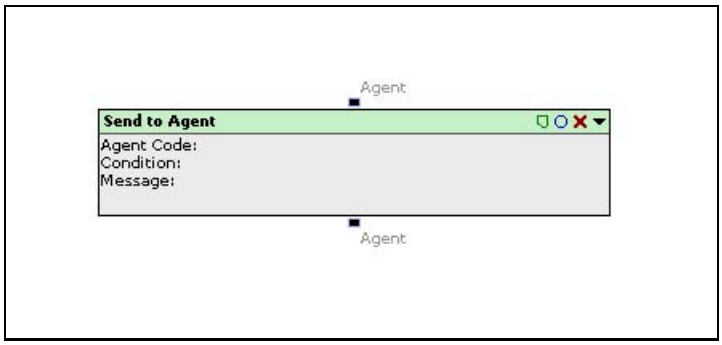

Figure 5.8. Send to Agent Node

2. Agent Code: It is the name of the receiver agent.
3. Condition: This field is for specifying in which condition the message will be sent.
4. Message: This field identifies contents of the message. The expression written here may be used to overwrite on the message text which is created in MEP. (The expression returns a message text as a string.) Thus, the content of the message to be transfered is finally determined here.

In ICRON, there is an Agent System, and all agents are connected to this agent system. The Agent System plays the major role to forward a message among the agents. A message sender agent find the receiver agent (respondent to the message) through the agent system by just specifying the type and the name of the receiver in Send to Agent node. If the condition expression written in the Send to Agent node returns true, then the message is send to the agent system with content specified in the message field. If there exist an agent with the specified type and name in the agent system, then the message is directed to the address of the respondent agent.

The Figure 5.9 is a declarative example for Dispatch Message Algorithm of agents. In dispatch messages algorithm, messages are branched according to message codes. Then by using Send to Agent node, the respondent is defined, the condition to send the message is specified, the message content may be reformed, and the message is directed to the address of the respondent agent. This algorithm encapsulates all possible messages that can be sent by a specific agent into a single description. It also specifies the agents to receive the resulting messages.

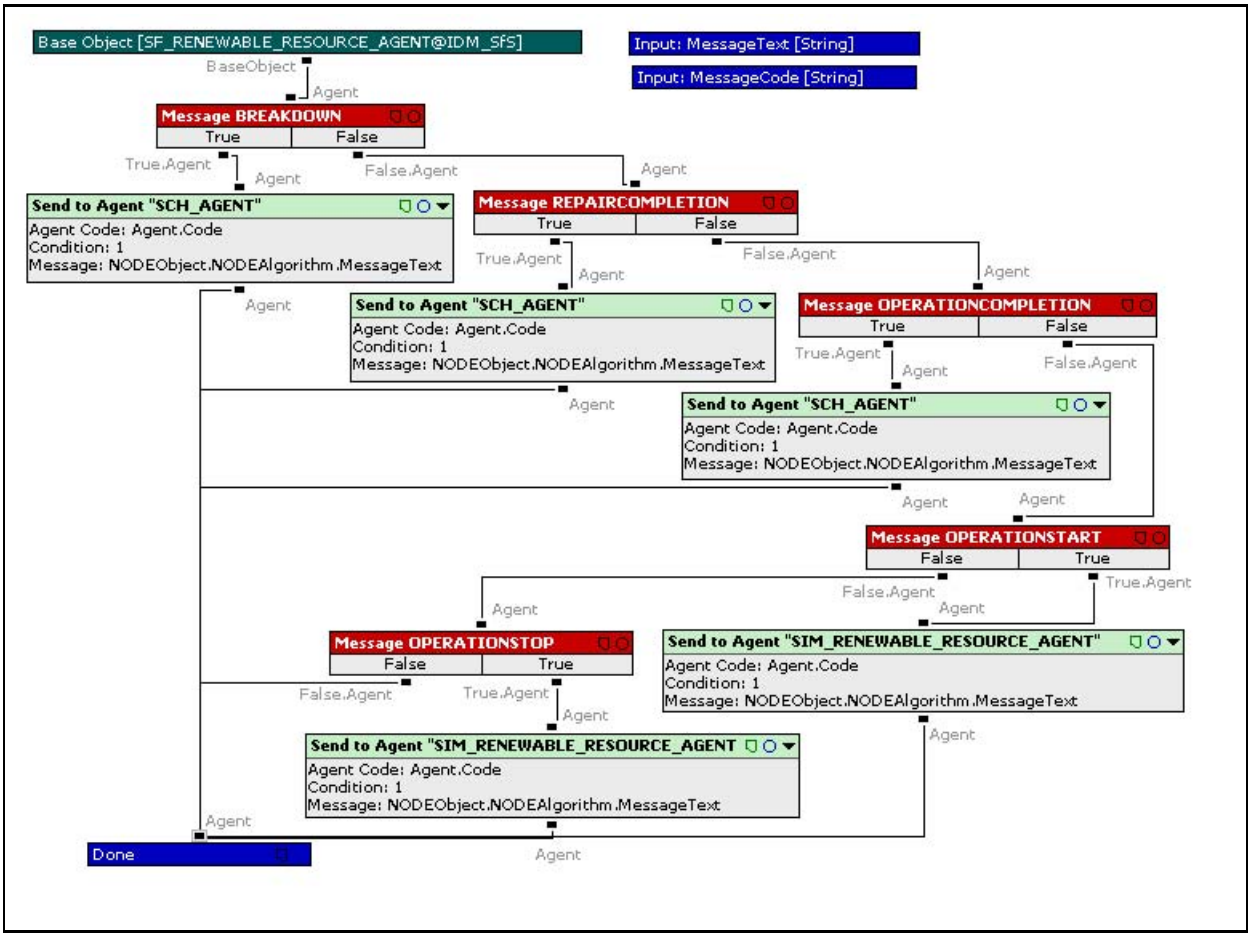

Figure 5.9. Dispatch Message Algorithm of agents

6. DISTRIBUTED SIMULATOR

In dynamic and stochastic manufacturing environments, a scheduling system should be able to modify its schedule to adapt to changing situations such as machine breakdowns, due-data changes, new rush orders, canceled orders and so on. In dealing with problems associated with these uncertainties, the shop-floor control system plays a very important role. In this chapter Distributed simulator (IDM) is introduced to control and monitor shop floor components and to coordinate the production activities. Simulation is an efficient tool used for modeling, analyzing and design of systems which enables testing different issues of the system.

IDM has three basic component: IDMSimulator, Shop Floor System, and Scheduling System. IDMSimulator represents the renewable resources (machines). Shop Floor System is called "IDMSfs" which consists of the real life data of the shop floor environment. Scheduling System is called "IDMSch" and it consists of scheduling agents which are decision making tools for scheduling problems.

- **IDMSimulator:** IDMSimulator represents the machines in the shop floor. IDMSimulator holds real life data of the renewable resources in the shop floor system by SIM-RENEWABLE-RESOURCE objects (i.e. machines). To represent real life, a machine does not know the specifications of the operations it process. Information provided by a machine are: whether the machine processing an operation or not, if it is processing an operation, then when the current operation was started, what is the speed of the processing, the machine is currently broken or not, if it is broken, then when it realized the breakdown.

  Ultimately we intended to build the simulator such that it can be replaced by a real manufacturing cell. In other words, instead of the simulator processes triggering of messages, the messages can be received even from various real manufacturing entities.
- **Shop Floor System / IDMSfs:** IDMSfs is the software component of a shop floor control system such as machine drivers or controllers. IDMSfs is capable

of composing this real life data with execution data of software applications. Renewable resources in a shop floor are physical entities that can provide very basic information about their situations i.e. data of machines.

These kind of software components combines the the real life data of machines with the data of other scheduling entities of shop floor environment. For example IDMSfs includes SF-RENEWABLE-RESOURCE objects to represents machines and SF-OPERATION objects that hold the dynamic shop floor execution data about each operation in the system such as batch size, the current resource, the last operation start time, percent completion of last unit etc. If a machine (SF-RENEWABLE-RESOURCE) is currently processing an operation, then it has a connection to its current operation object.

- **Scheduling System / IDMSch:** IDMSch consists of a SCHEDULE AGENT which is the central scheduler, a SCH-JOB AGENTs which manages predecessor - successor relations of operations, and SCH-RENEWABLE-RESOURCE AGENTs which are local schedulers. These agents make scheduling decisions by their individual computing facilities based on their local information and local decision making protocol (Local/global scheduling algorithms and scheduling trigger protocols).

In summary changing situations are represented by a simulator (i.e. IDMSimulator) and provided to a system which represents the execution of the shop floor (i.e. IDMSfs). Shop floor execution data is represented in a data base and it is dynamically updated by IDMSfs system. The decision makers (i.e. scheduling agents) load the data or part of the data about the instance of the problem which they solve, to their individual domains and it is the simulation of the dynamic environment.

The simulation process is built on a time driven discrete event based mechanism. A chain of simulation events are created then stored in an simulation event queue and realized respectively as time goes. Simulation time is an instant of time in a simulation run and is a scaled factor of the real time. To give an example, one real second may correspond to one weak of the simulator time (if each time tick of the program takes one second). Thus simulation system advances with constant simulation time blocks.

Each simulation event has a time attribute (with respect to simulation time unit) which denotes the time that event will occur. The simulation events which have occurrence times less than or equal to the simulation time, are evaluated in these time blocks as time goes.

Simulation events of IDMSimulator are defined as classes in simulator package as shown in Figure 6.1.

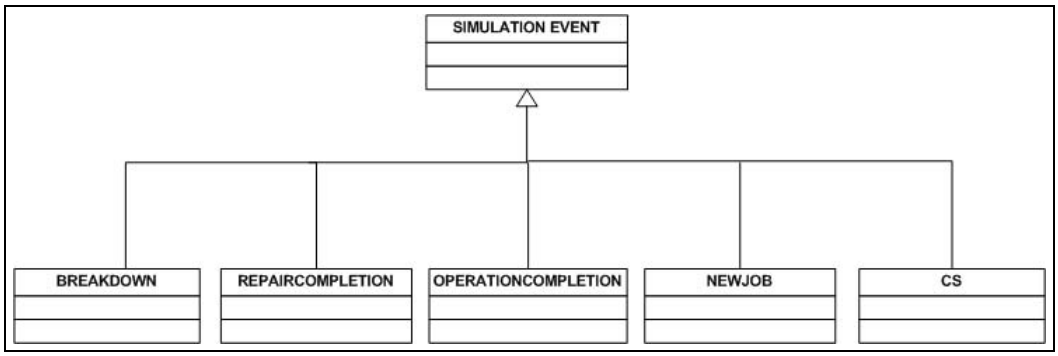

Figure 6.1. Simulation Event Class Diagram

Considering the Figure 6.1 there are five main simulation events in IDMSimulator:

1. Central Schedule event (CS), when simulator force the system to obtain a new central schedule.
2. New Job event (NEWJOB), when a new job is introduced to the system.
3. Breakdown event (BREAKDOWN), when one of the machine is broken.
4. Repair Completion event (REPAIRCOMPLETION), when the broken machine is repaired.
5. Operation Completion event (OPERATIONCOMPLETION), when one of the operations is completed for a machine.

The simulator arbitrarily generates the simulation events including breakdown event of machines, new job arrival event and central schedule event in order to symbolize changing situations. Breakdown events results creation of repair completion events. Similarly, when a machine starts to process an operation with respect to the direction of a dispatcher, corresponding operation completion event is also created.

The simulation events which are given in Figure 6.1 and explained briefly above, result messaging among related agents. Simulation events are not necessarily similar to those that are created by MEPs (as a consequence of interpretation of system messages). Namely in these messaging flows, system messages are either emanate from simulation events or from processing of agents. Distributed running mechanism of IDM is based on messaging flows among system agents and is shown in Figure 6.2.

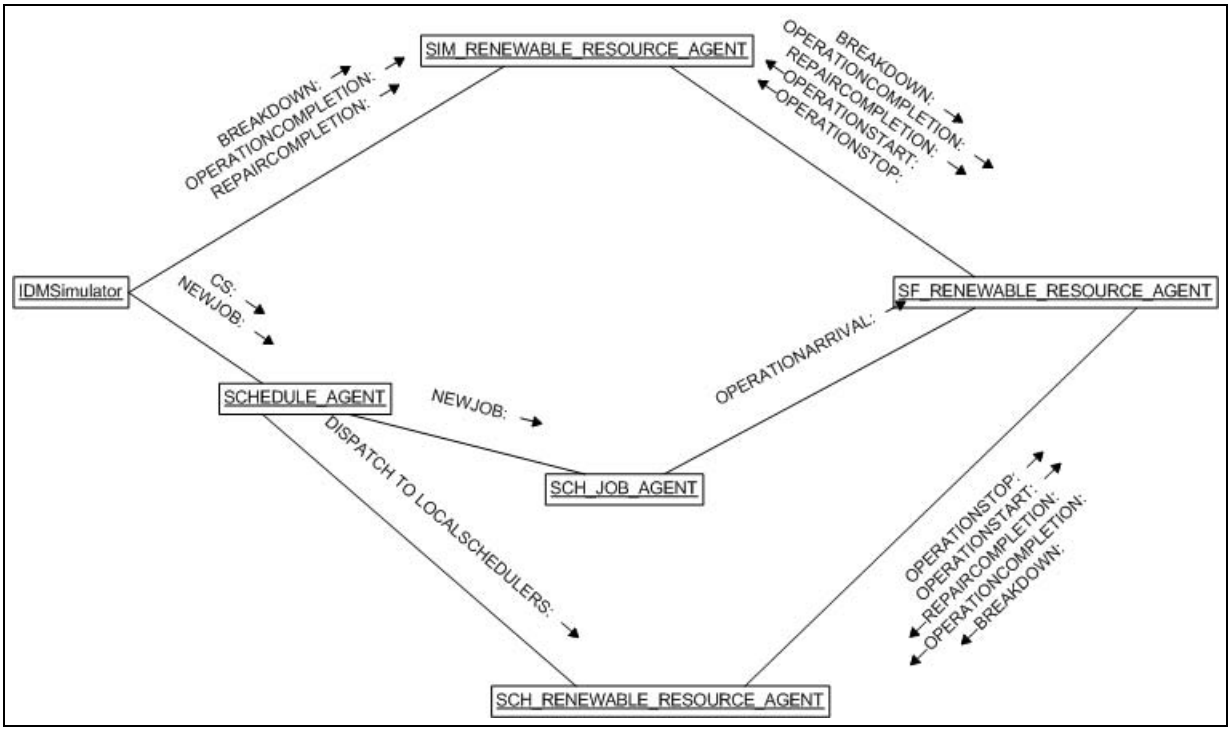

Figure 6.2. Collaboration Diagram among IDM agents

As shown in Figure 6.2 there are five types of agents in IDM: SIM-RENEWABLE-RESOURCE-AGENT, SF-RENEWABLE-RESOURCE-AGENT, SCH-RENEWABLE-RESOURCE-AGENT, SCH-JOB-AGENT, and SCHEDULE-AGENT. Messaging flows among those agents are resulted from the simulation event triggers.

A SIM-RENEWABLE-RESOURCE-AGENT belongs to IDMSimulator system and is associated with SIM-RENEWABLE-RESOURCE objects (i.e. machines). Our proposed model promises that real shop floor entities can be replaced by these virtual SIM-RENEWABLE-RESOURCE objects and SIM-RENEWABLE-RESOURCE-AGENT can manage message processing on the data obtained from real life.

SF-RENEWABLE-RESOURCE-AGENT belongs to IDMSfs and manages the issues associated with SF-RENEWABLE-RESOURCE objects (to represent machines as softwares) and SF-OPERATION objects in shop floor software. This kind of an agent holds a machine object as its local object, and provides a connection to the current operation object on that machine through this machine object.

As mentioned above SCH-RENEWABLE-RESOURCE-AGENT is the local scheduler (in particular the dispatcher) for a machine. Thus the local object assigned to this agent is also a machine (SCH-RENEWABLE-RESOURCE object) which represent the respective machine in IDMSch domain. For example considering the collaboration diagram in Figure 6.2 again, when a SCH-RENEWABLE-RESOURCE-AGENT receives one of the repair completion or operation completion messages, related to its local machine object, it makes the decision of which operation should be start (or stop) among the operations in list on hand.

The messaging flows due to BREAKDOWN, REPAIRCOMPLETION, and OPERATIONCOMPLETION simulation events which can be followed from the collaboration diagram are based on a very similar logic until the messages reaches to the corresponding SCH-RENEWABLE-RESOURCE-AGENT. Local Rescheduling protocol is determined in the MEP of SCH-RENEWABLE-RESOURCE-AGENT. In this chapter massaging flows for BREAKDOWN, local rescheduling process triggering and central rescheduling process triggering are explained in detail.

6.1. Messaging flow for BREAKDOWN simulation event

In this section massaging flow for BREAKDOWN simulation event is given in detail, then the massaging flows for REPAIRCOMPLETION and OPERATIONCOMPLETION simulation events are explained in subsections by referring to the figures given in appendicies.

IDMSimulator triggers a BREAKDOWN simulation event which is related to a specific machine in the shop floor. As seen from the collaboration diagram given in Fig-

ure 6.1 it arrives to the SIM-RENEWABLE-RESOURCE-AGENT, i.e BREAKDOWN message is send to this agent. When SIM-RENEWABLE-RESOURCE-AGENT receives a message, the messages triggers handle message algorithm of the agent, thus agent finds the corresponding MEP (BREAKDOWN MEP) according to the context of the message by its handle message algorithm.

BREAKDOWN MEP of SIM-RENEWABLE-RESOURCE-AGENT is shown in the Figure 6.3. As shown in the Figure 6.3, this MEP interprets the message as BREAKDOWN event and triggers the state chart with state chart code "SIM-RR" and then provide the BREAKDOWN message to the "Dispatch Message" algorithm of SIM-RENEWABLE-RESOURCE-AGENT.

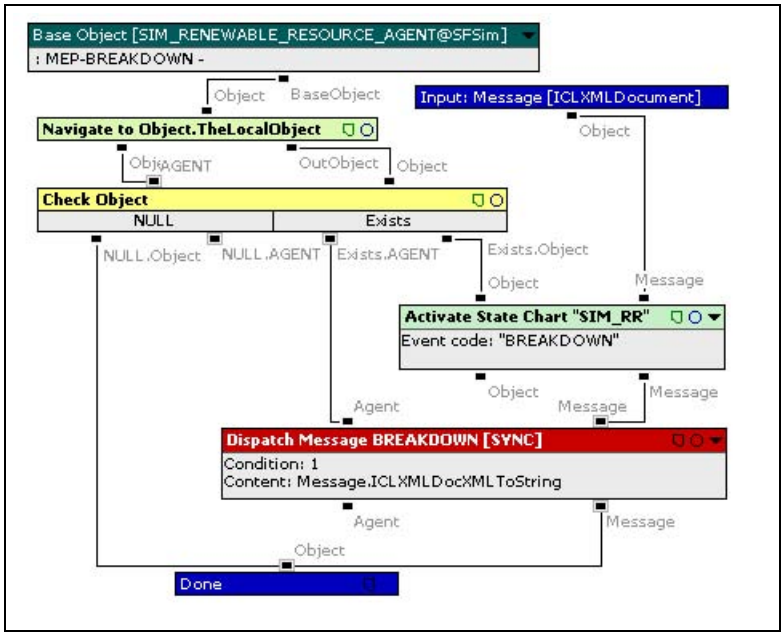

Figure 6.3. MEP of SIM-RENEWABLE-RESOURCE-AGENT for breakdown message

The Dispatch Message Algorithm of SIM-RENEWABLE-RESOURCE-AGENT is shown in the Figure 6.4.

By following the branch in Figure 6.4, SIM-RENEWABLE-RESOURCE-AGENT sends the message to an agent in a type of SF-RENEWABLE-RESOURCE-AGENT with the name of the current agent. (Note that each agent holds a machine object as its local object in each domain, and the name of the machine is also the name of

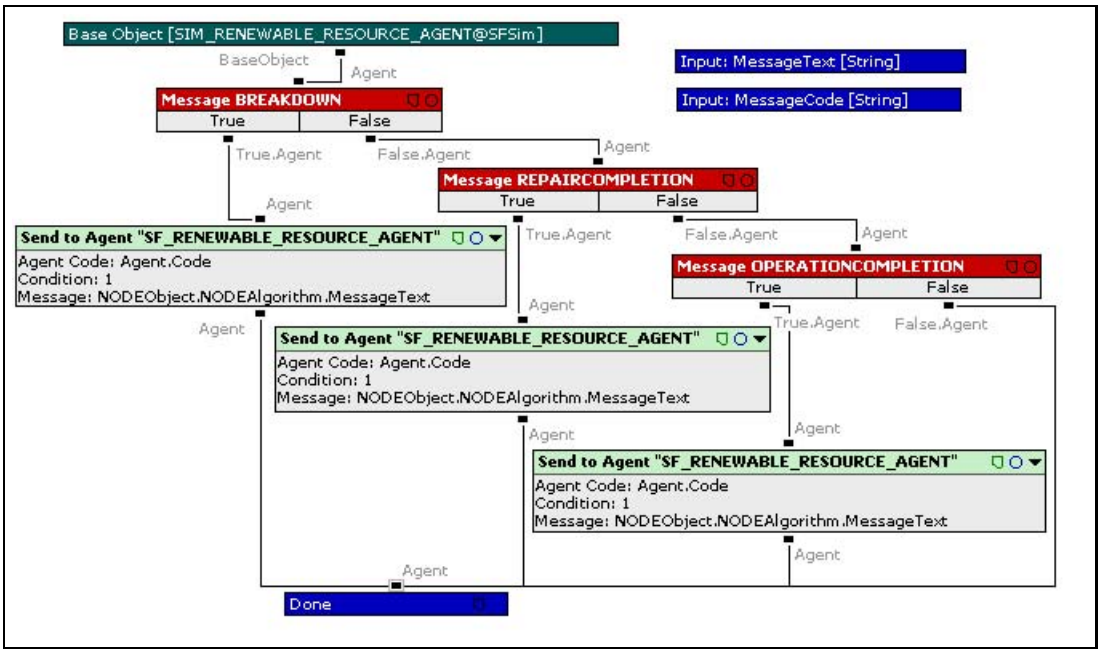

Figure 6.4. Dispatch Message Algorithm of SIM-RENEWABLE-RESOURCE-AGENT

the agent.) Thus the BREAKDOWN is sent to the corresponding SF-RENEWABLE-RESOURCE-AGENT.

Similarly when SF-RENEWABLE-RESOURCE-AGENT receives a message, it finds the corresponding MEP (BREAKDOWN MEP)according to the context of the message. BREAKDOWN MEP of SF-RENEWABLE-RESOURCE-AGENT is explained in "MEP as an Algorithm" section of Chapter 5.

Referring to Figure 5.7 again, the message is interpreted as "BREAKDOWN" event for the machine, as "OPERATION STOP" event for the operation and as "UPDATE DB" (update data base) event for data manipulations of both operation and machine objects. After processing of the BREAKDOWN message, it is given to the "Dispatch Message" algorithm of SF-RENEWABLE-RESOURCE-AGENT. The Dispatch Message Algorithm of SF-RENEWABLE-RESOURCE-AGENT is given in the Figure 6.5.

When SCH-RENEWABLE-RESOURCE-AGENT receives the BREAKDOWN message, the corresponding MEP processes the message in order to represent the cur-

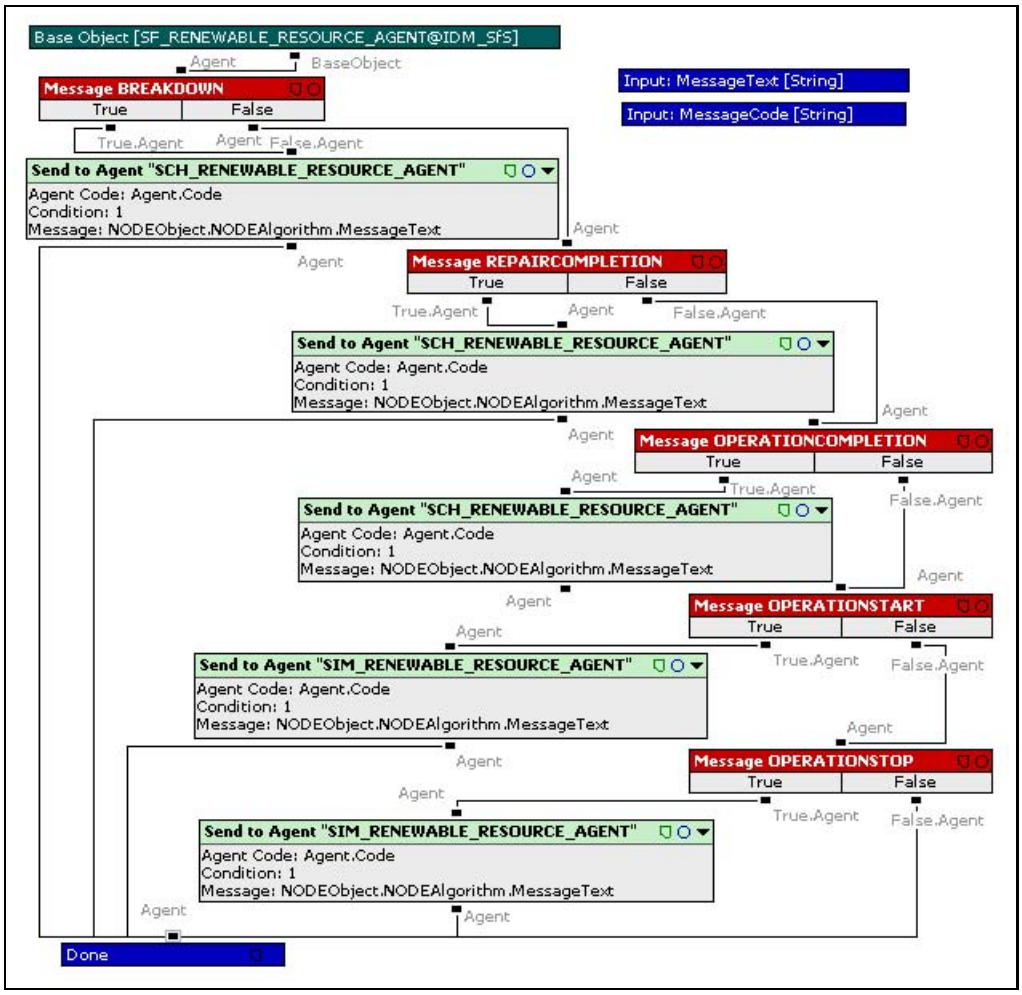

Figure 6.5. Dispatch Message Algorithm of SF-RENEWABLE-RESOURCE-AGENT

rent situation of the machine in IDMSch and to make a decision with respect to the message. For example whether it can create a new schedule for the machine (in order to provide to the machine when repair completion occurs) or it can wait for the repair completion event to create the recent schedule.

6.1.1. Messaging flow for REPAIRCOMPLETION simulation event

IDMSimulator triggers a REPAIRCOMPLETION simulation event which is related to a specific machine in the shop floor. Then it arrives to the SIM-RENEWABLE-RESOURCE-AGENT. When SIM-RENEWABLE-RESOURCE-AGENT receives the message, it finds the corresponding MEP (REPAIRCOMPLETION MEP) according to the context of the message by its handle message algorithm.

REPAIRCOMPLETION MEP of SIM-RENEWABLE-RESOURCE-AGENT is shown in Figure A.1. This MEP interprets the message as "REPAIR-COMPLETED" event and triggers the state chart with state chart code "SIM-RR" and then provide the REPAIRCOMPLETION message to the "Dispatch Message" algorithm of SIM-RENEWABLE-RESOURCE-AGENT. (The Dispatch Message Algorithm of SIM-RENEWABLE-RESOURCE-AGENT is shown in Figure 6.4.)

Thus REPAIRCOMPLETION is sent to the corresponding SF-RENEWABLE-RESOURCE-AGENT. When SF-RENEWABLE-RESOURCE-AGENT receives the message, it provides the message to the REPAIRCOMPLETION MEP of SF-RENEWABLE-RESOURCE-AGENT. REPAIRCOMPLETION MEP is presented in Figure B.1.

After processing of the REPAIRCOMPLETION message, it is given to the "Dispatch Message" algorithm of SF-RENEWABLE-RESOURCE-AGENT. (The Dispatch Message Algorithm of SF-RENEWABLE-RESOURCE-AGENT is given in Figure 6.5.) Then the SCH-RENEWABLE-RESOURCE-AGENT receives the REPAIRCOMPLETION message.

6.1.2. Messaging flow for OPERATIONCOMPLETION simulation event

IDMSimulator triggers a OPERATIONCOMPLETION simulation event which is related to a specific machine in the shop floor. Then it arrives to the SIM-RENEWABLE-RESOURCE-AGENT. When SIM-RENEWABLE-RESOURCE-AGENT receives the message, it finds the corresponding MEP (OPERATIONCOMPLETION MEP) according to the context of the message by its handle message algorithm.

OPERATIONCOMPLETION MEP of SIM-RENEWABLE-RESOURCE-AGENT is shown in Figure A.2. This MEP interprets the message as "OPERATIONCOMPLETION" event and triggers the state chart with state chart code "SIM-RR" and then provide the OPERATIONCOMPLETION message to the "Dispatch Message" algorithm of SIM-RENEWABLE-RESOURCE-AGENT. (The Dispatch Message Algorithm of SIM-RENEWABLE-RESOURCE-AGENT is shown in Figure 6.4.)

Thus the message OPERATIONCOMPLETION is sent to the corresponding SF-RENEWABLE-RESOURCE-AGENT. When SF-RENEWABLE-RESOURCE-AGENT receives the message, it provides the message to the OPERATIONCOMPLETION MEP of SF-RENEWABLE-RESOURCE-AGENT. OPERATIONCOMPLETION MEP of SF-RENEWABLE-RESOURCE-AGENT is presented in Figure B.2.

After processing of the OPERATIONCOMPLETION message, it is given to the "Dispatch Message" algorithm of SF-RENEWABLE-RESOURCE-AGENT . (The Dispatch Message Algorithm of SF-RENEWABLE-RESOURCE-AGENT is given in Figure 6.5.) Then the SCH-RENEWABLE-RESOURCE-AGENT receives the OPERATIONCOMPLETION message.

6.2. Local Rescheduling Process

SCH-RENEWABLE-RESOURCE-AGENT receives one of the BREAKDOWN, REPAIRCOMPLETION, and OPERATIONCOMPLETION messages , the corresponding MEP shown in Figure 6.6 processes the message.

Following the figure, this agent firstly updates the data by activating the state chart named SCH-RR-DB code (i.e state chart of data base aspect) with "SYNCHRONZE" event. Thus this agent now knows the current situation of the machine (i.e. the current situation of the machine is represented in IDMSch domain). Considering Figure 6.6, local rescheduling process is triggered with one of the BREAKDOWN, REPAIRCOMPLETION, and OPERATIONCOMPLETION messages which is interpreted as "STARTSCHEDULE" event. Namely the SCH-RR state chart is activated with "STARTSCHEDULE" event (local scheduling algorithm is defined in the action in SCH-RR state chart).

Obviously if the BREAKDOWN event occurs, this agent will wait for REPAIRCOMPLETION event to direct the machine due to the recent schedule on hand. To give the example of how rescheduling process triggering is handled, suppose that one of the REPAIRCOMPLETION or OPERATIONCOMPLETION event is realized. After

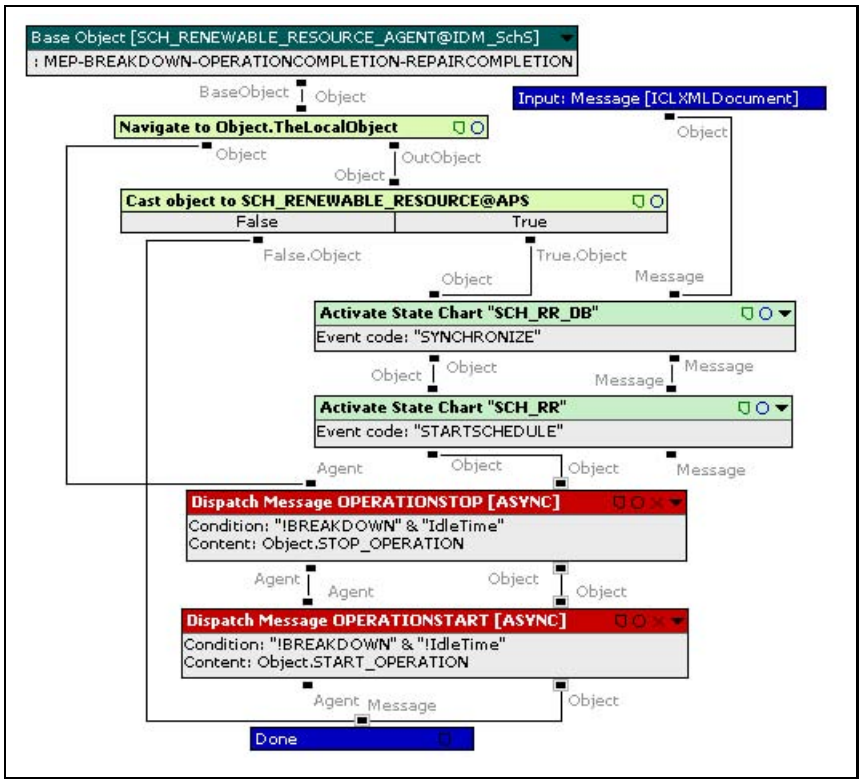

Figure 6.6. BREAKDOWN / REPAIRCOMPLETION / OPERATIONCOMPLETION MEP of SCH-RENEWABLE-RESOURCE-AGENT

activating the state charts SCH-RR-DB and SCH-RR, a recent schedule is obtained. Corresponding to the schedule on hand, either the machine is started with an operation or it is stopped until the realization of some conditions. Thus one of the OPERATIONSTOP or OPERATIONSTART messages is created and provided to the Dispatch Message algorithm of SCH-RENEWABLE-RESOURCE-AGENT shown in Figure 6.7.

Suppose that the decision maker wants to start an operation, then OPERATIONSTART message is send to the SF-RENEWABLE-RESOURCE-AGENT as can be seen from Figure 6.7. Then SF-RENEWABLE-RESOURCE-AGENT supplies the message to its corresponding OPERATIONSTART MEP which is shown in Figure 6.8.

After processing of the OPERATIONSTART message, it is given to the "Dispatch Message" algorithm of SF-RENEWABLE-RESOURCE-AGENT. As can be followed from Figure 6.5, the respondent agent to this message is SIM-RENEWABLE-RESOURCE-AGENT. In other words, the decision is realized by the software and now

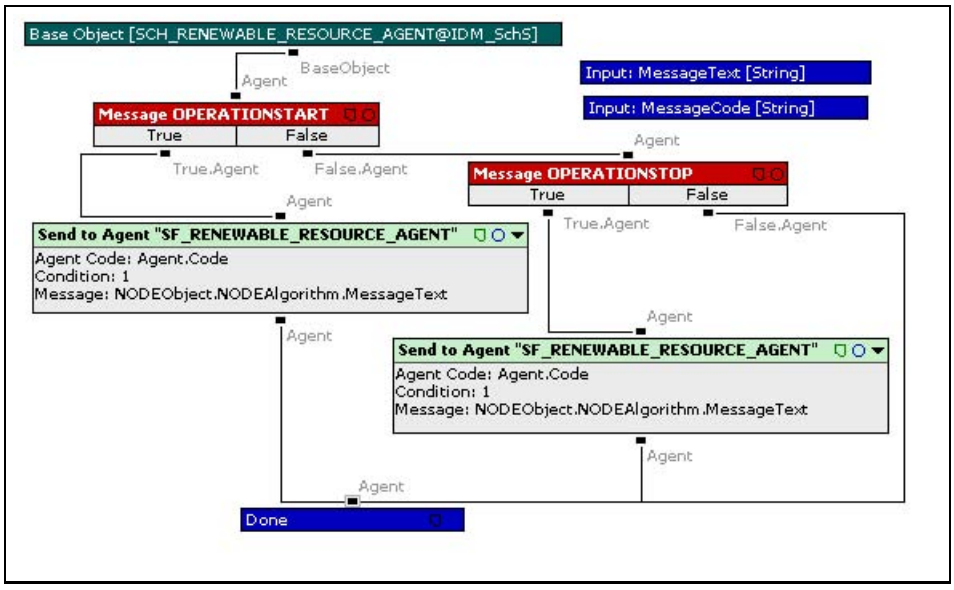

Figure 6.7. Dispatch Message Algorithm of SCH-RENEWABLE-RESOURCE-AGENT

it should be applied by the real machine.

OPERATIONSTART MEP of SIM-RENEWABLE-RESOURCE-AGENT is shown in the Figure 6.9.

6.3. Messaging flow for Central Schedule (CS) simulation event - Central Rescheduling Process

IDMSimulator triggers CS simulation event which can be created arbitrarily or in some determined frequencies. As can be seen from the collaboration diagram in Figure 6.2, CS simulation event arrives as CS message to the SCHEDULE-AGENT which is the central scheduler. Corresponding CS MEP of SCHEDULE-AGENT is shown in Figure 6.10.

As in Figure 6.10 SCHEDULE state chart is activated with START-SCHEDULE event (global scheduling algorithm is defined in the action of SCHEDULE state chart), then the message "Dispatch-To-Local-Schedulers" is created and given to the dispatch message algorithm of the SCHEDULE-AGENT in Figure 6.11.

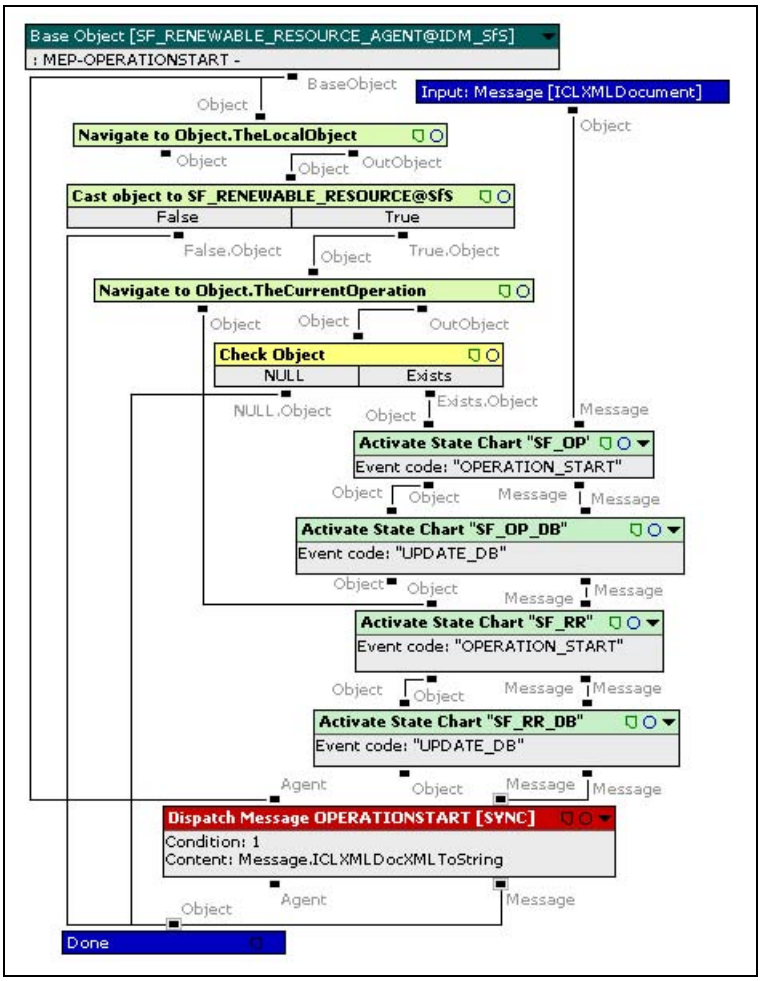

Figure 6.8. MEP of SF-RENEWABLE-RESOURCE-AGENT for operation start message from SF-RENEWABLE-RESOURCE-AGENT

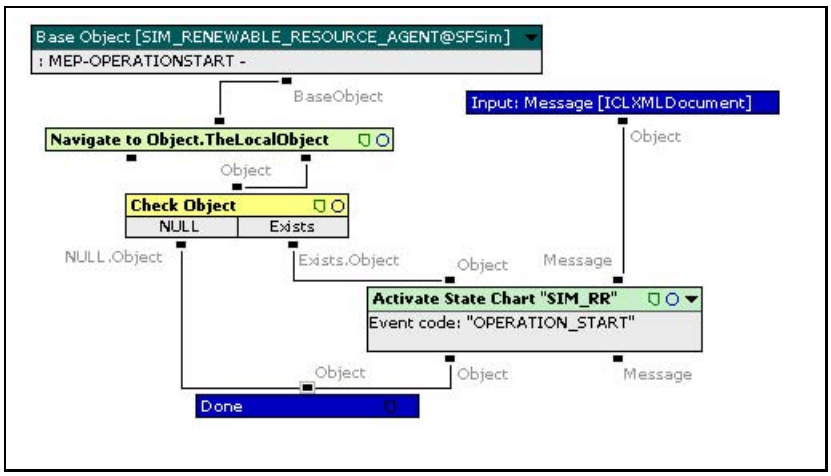

Figure 6.9. MEP of SIM-RENEWABLE-RESOURCE-AGENT for operation start message

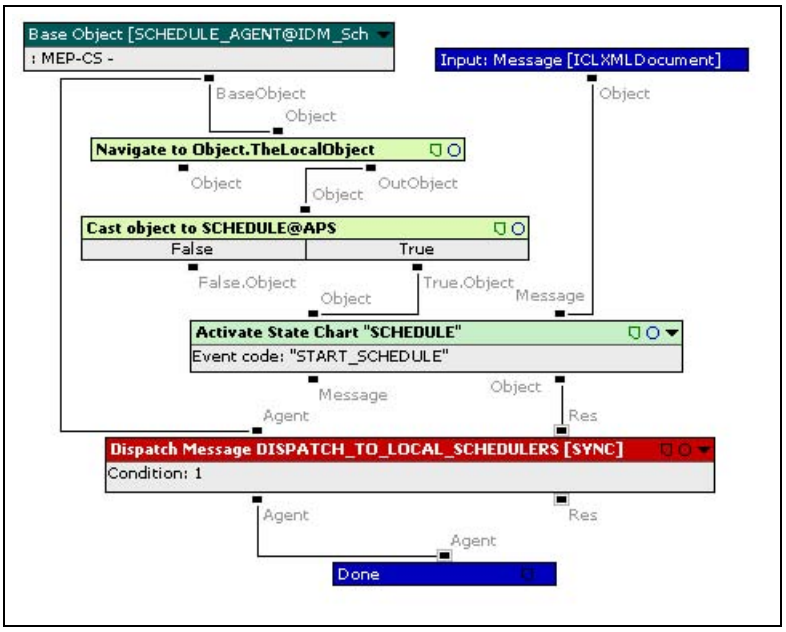

Figure 6.10. MEP of SCHEDULE-AGENT for central schedule message

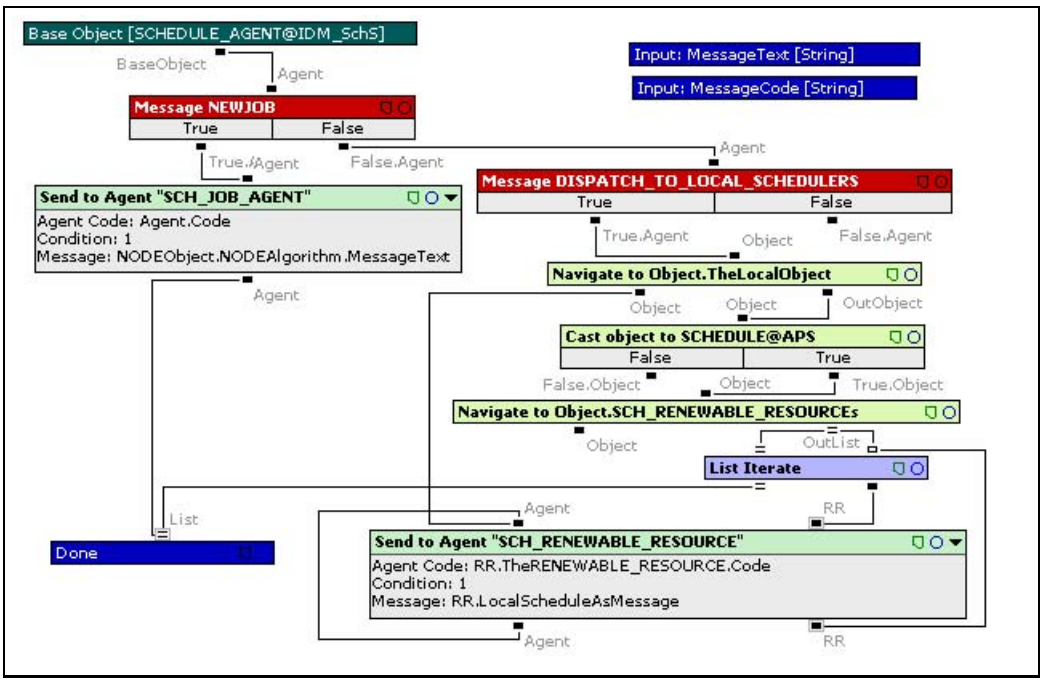

Figure 6.11. Dispatch Message Algorithm of SCHEDULE-AGENT

The SCHEDULE-AGENT is related with a schedule object. The schedule is decomposed into various local schedule objects i.e. SCH-RENEWABLE-RESOURCEs. As can be seen from Figure 6.11, SCHEDULE-AGENT navigates to respective SCH-RENEWABLE-RESOURCEs through its local schedule object. The message is braked into pieces and distributed to the corresponding SCH-RENEWABLE-RESOURCE-AGENTs in "Send to Agent Node" iteratively. Thereby, each SCH-RENEWABLE-

RESOURCE-AGENT receives a message named DISPACT-TO-LOCAL-SCHEDULERS.

The corresponding MEP to handle this message is shown in Figure 6.12.

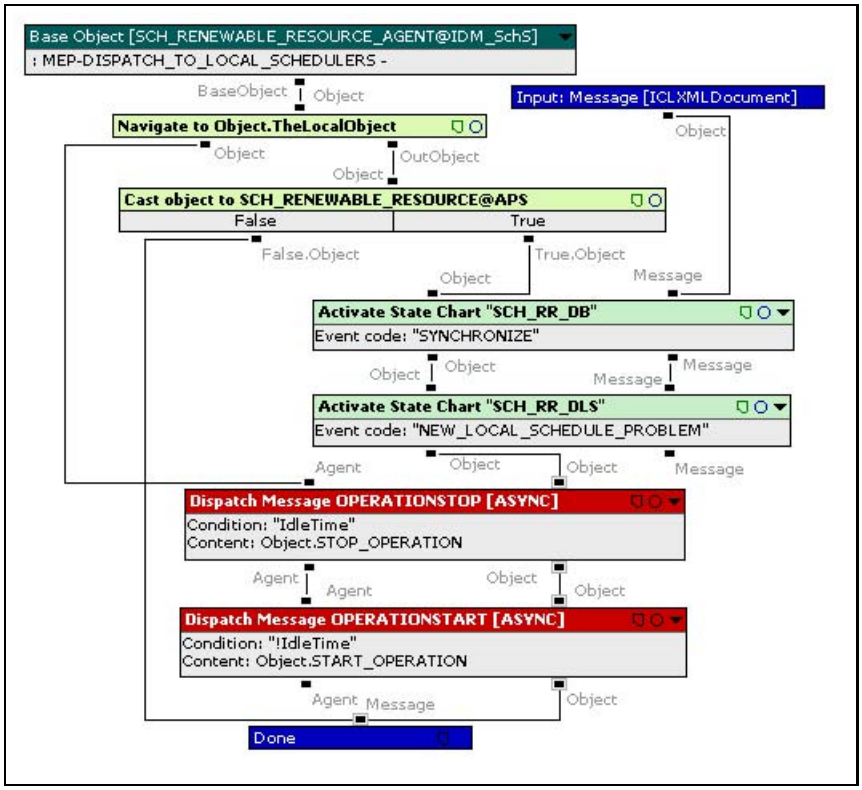

Figure 6.12. MEP of SCH-RENEWABLE-RESOURCE-AGENT for DISPACT-TO-LOCAL-SCHEDULERS message

In short, the DISPACT-TO-LOCAL-SCHEDULERS MEP of SCH-RENEWABLE-RESOURCE-AGENT firstly updates the data by activating the state chart named SCH-RR-DB code with “SYNCHRONZE” event. Then , the SCH-RR-DLS state chart is activated with “NEW-LOCAL-SCHEDULE-PROBLEM” event (local scheduling algorithm is defined in the action in SCH-RR state chart).(Figure 6.12)

Corresponding to the schedule on hand, either the machine is started with an operation or it is stopped until the realization of some conditions. Thus one of the OPERATIONSTOP or OPERATIONSTART messages is created and provided to the Dispatch Message algorithm of SCH-RENEWABLE-RESOURCE-AGENT. This process is similar to the process explained as in Local Rescheduling Process section.

7. DISCUSSIONS

The proposed framework is based on state representation of objects in state charts. State charts are defined as algorithms and verification of execution on these algorithms is manipulated as follows: After a state chart is activated with an event, if there is an action to execute, then it is expected that the object will return its current state as the state that follows the activated action node. So when activate state chart node executes a state change, resulting state of the object can be checked by evaluation of the condition function of the state node.

In this study, the aim is to assess the issues which are expressed in Chapter 3. The first issue is providing the consistency of the data within an object by preventing the access of multiple transformations on the same data. Each transformation that an object performs in respect to a certain aspect is classified as action in a state chart. Actions are executed in particular conditions. Namely, if the object is currently available to perform an action and if the action triggering event is related with the current state of the object, then the transformation on the data is performed. State nodes denote whether the object is stable or not (whether the object is ready to perform a process or not).

For some actions object does not necessarily wait the end of its action in order to accept a similar event to handle. However, if the conditional expression in the state node is related with some attributes which are transformed during the execution of the action node, then the object will be in unstable state. So, during the execution, the object is unavailable to perform another action in the same state chart. In this kind of state charts, some mutual exclusion algorithms are needed. These locking algorithms can be provided in such a way that object performs one possible action at a time, and can be introduced to relevant state charts by identifying the corresponding object as being stable or unstable. The proposed model reduces the complexity of these locking algorithms since the event nodes already act like conditional branching and also the conditional expressions which are defined in the state nodes support to detect points

where the locking algorithms are required.

The proposed model provides the efficiency that accesses to the shared data within an object due to some transformations are limited by separating the aspect of objects in distinct state charts. Thus independent activities are prevented to be mixed in a single state chart as a cross product, because transformations which are related with distinct part of the data (within an object) are defined in other state charts. Consequently, the multiple processes (which are interpreted as events in state charts) may trigger a transformation on some part of the shared data, but now these transformations (which are easily detectable) are related with as small portion of the data as possible.

As mentioned previously, each object may have many aspects related to the scheduling, data and communication. Parallel state charts may be specific to an object or may be not specific to a single object but they intersect many modules in a system. The later class of parallel state charts provides systematic identification, separation, representation of specific concerns in a place. Thus localization is supported, and results in reducing development and maintenance tasks.

At this point, the second important problem is discussed which is the consistency of the state of shared data among related agents. In the proposed model the data manipulation concerns are defined in database state charts. As mentioned previously the object either reads the data from data base by triggering "synchronize" event or writes its current data on the related part of the database by triggering "update-database" event. The proposed model supports the monitoring of these event triggering processes. For example, consider that some part of the data in database is shared by various objects, since these objects are in limited number, each can easily be determined, and whenever the data is changed in database due to one of these objects activities (i.e. it is updated), the database state charts of other corresponding objects can be triggered by synchronize events in order to provide accurate representations of local data.

8. CONCLUSIONS

The main aim of the study is to provide an accurate representation of the dynamic environment in which agents operate. The proposed framework is based on the state representation of agents and objects controlled by agents. Each object may have many aspects related to scheduling, data, and communication. Each aspect is modeled in a distinct state chart. A state chart defined for an object represents the possible states of the object, events that this object may need to respond in the form of actions, and possible state changes as a result of these actions. State of an agent is the collection of states of the objects that it controls in all aspects.

In this distributed system, objects need to coordinate their activities with other objects. Coordination and communication are provided by agents. Agents keep objects away from messaging issues and objects do not communicate without an agent connection. Each agent may be associated with a class of objects or a group of classes of objects. Communication between agents are managed through messaging.

Messages received by agents are interpreted by MEP definitions. MEP first decides which objects are related to the message. Then it maps the message to a respective event for the object, and triggers the state chart of the related aspect. Finally, MEP may create messages for other agents, these messages are dispatched by agent. In this framework, event is an interpretation of a received message in the context of a state chart, thus by using the proposed mechanism, messages are allowed to be reinterpreted in different aspects as corresponding events and dependencies between state charts are reduced.

Through this framework, possible states of all objects in the system within the scheduling process are defined. The distribution methodology is represented as state charts and MEPs. Scheduling process triggering protocols are defined through MEPs. Local / global scheduling algorithms are the actions in the state charts. Proposed state based modeling approach clearly represents the workings of a scheduling agent so that

effect of choices made in distributing the problem in terms of the overall performance of the scheduling methodology can formally be defined and analyzed.

The proposed model promises to be a framework to properly identify conditions and timing of possible inconsistencies and inaccuracy, and analyze the effectiveness of various distribution mechanisms in terms of scheduling performance.

9. FUTURE WORK

Distributing the scheduling results conflicts due to different scheduling policies among scheduling agents that have coupling constraints. This particular problem is mentioned as the third problem in Chapter 3. More precisely local scheduler agents may have distinct decisions on the same shared resource. An efficient handling mechanism should be provided for coupling constraints between agents.

If this kind of conflict is possible to be defined as the states of corresponding agents, then efficient conflict resolution function should be designed as actions for related agents. In other words, the conflict is identified as various states and for each state a different strategy is applied to solve the conflict. To give an example, consider two schedule related to a machine with different operation sequences, if both schedules are non- delay, then define the most critical operation (due to some criteria) and start processing; or if one schedule proposes to wait for a while, then evaluate the reasoning, etc.

As another future work, optimum frequency of re-scheduling for schedule agent may be determined by providing various scheduling process triggering protocols. This can be achieved by event processing among MEPs. For example when a message of a machine breakdown arrives, MEP can either immediately reschedule or wait for the repair completion. Moreover it can be provided that MEP ignores the breakdown of a single machine but it responds to the breakdown events of various machine (for example breakdown events of more than two similar machines). Event processing provides to analyze various cases.

Last but not least, the effectiveness of various negotiation mechanisms which are proposed in the literature can be analyzed. Negotiation mechanisms mostly force the system to act as it is centralized. The overall behavior of the system can be classified as distributed and centralized with respect to the state of agents, and the time consumed for each case can be observed for a specific scheduling problem.

APPENDIX A: SIM-RENEWABLE-RESOURCE-AGENT

REPAIRCOMPLETION MEP of SIM-RENEWABLE-RESOURCE-AGENT is shown in Figure A.1.

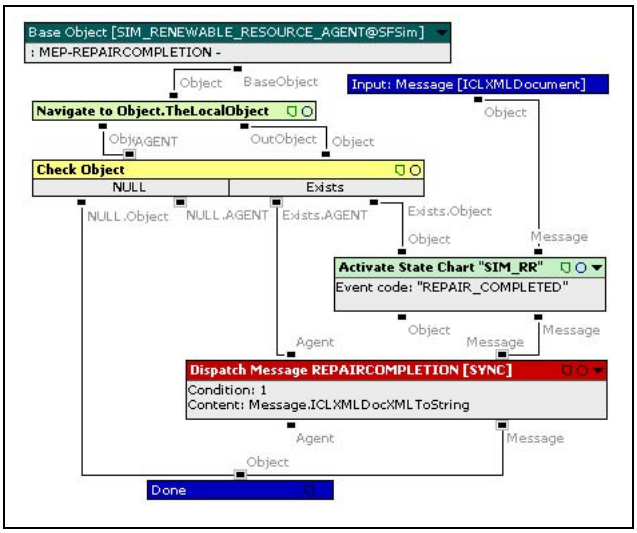

Figure A.1. MEP of SIM-RENEWABLE-RESOURCE-AGENT for repair completion message

OPERATIONCOMPLETION MEP of SIM-RENEWABLE-RESOURCE-AGENT is shown in Figure A.2.

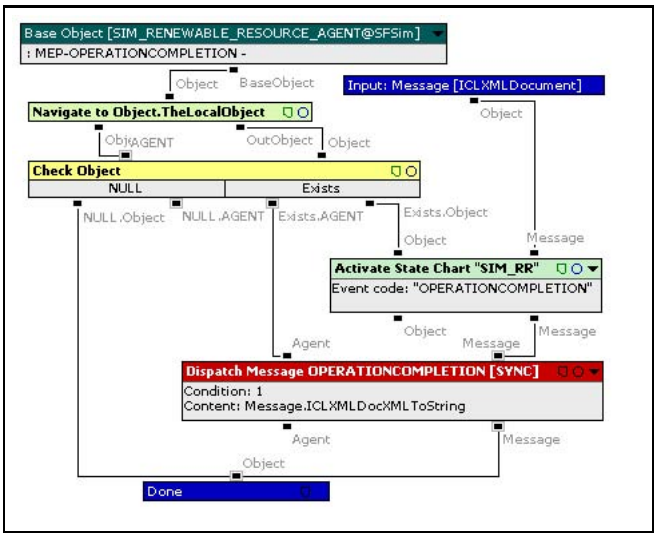

Figure A.2. MEP of SIM-RENEWABLE-RESOURCE-AGENT for operation completion message

OPERATIONSTOP MEP of SIM-RENEWABLE-RESOURCE-AGENT is shown in Figure A.3.

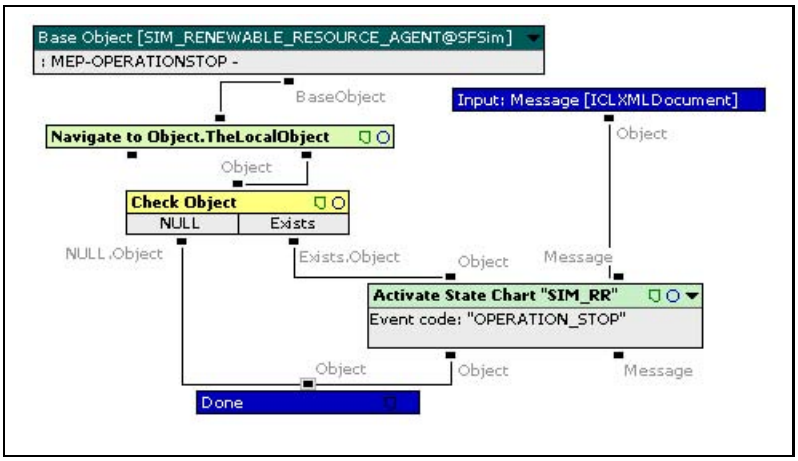

Figure A.3. MEP of SIM-RENEWABLE-RESOURCE-AGENT for operation stop message

APPENDIX B: SF-RENEWABLE-RESOURCE-AGENT

REPAIRCOMPLETION MEP of SF-RENEWABLE-RESOURCE-AGENT is shown in Figure B.1.

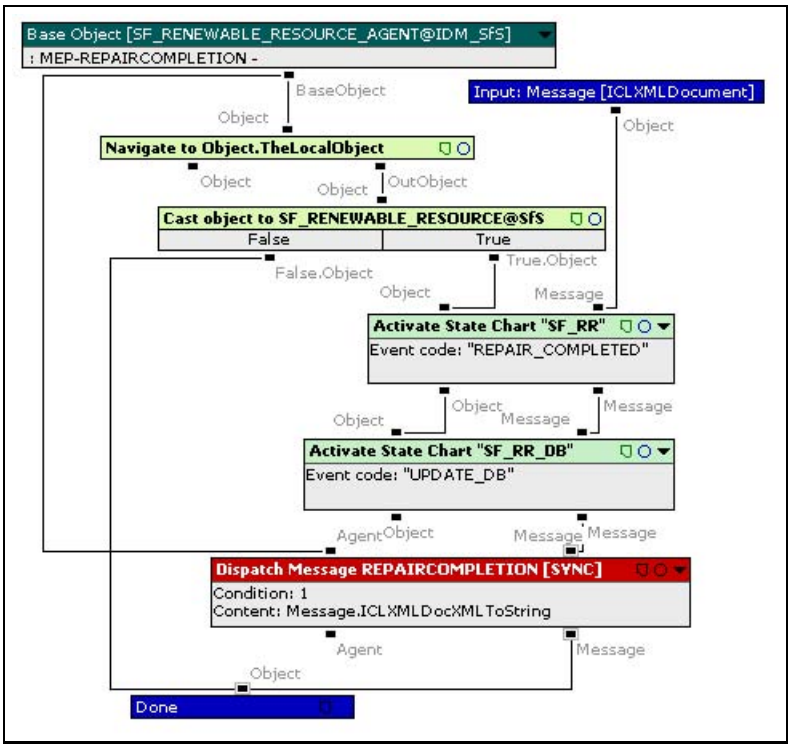

Figure B.1. MEP of SF-RENEWABLE-RESOURCE-AGENT for repair completion message

OPERATIONCOMPLETION MEP of SF-RENEWABLE-RESOURCE-AGENT is shown in Figure B.2.

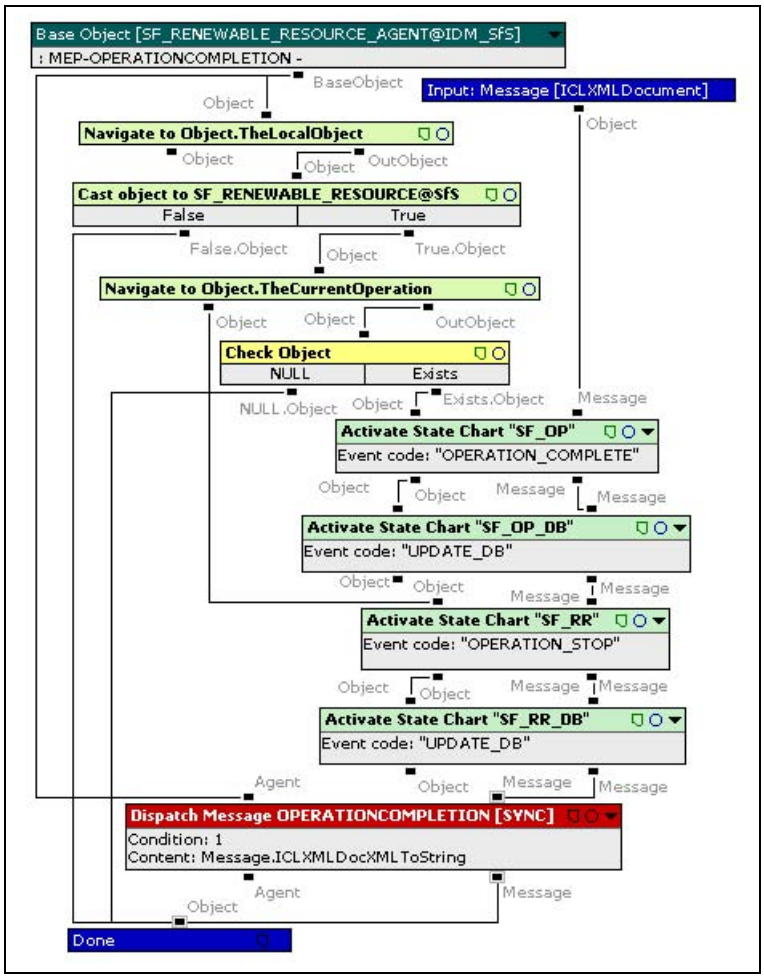

Figure B.2. MEP of SF-RENEWABLE-RESOURCE-AGENT for operation completion message

OPERATIONSTOP MEP of SF-RENEWABLE-RESOURCE-AGENT is shown in Figure B.3.

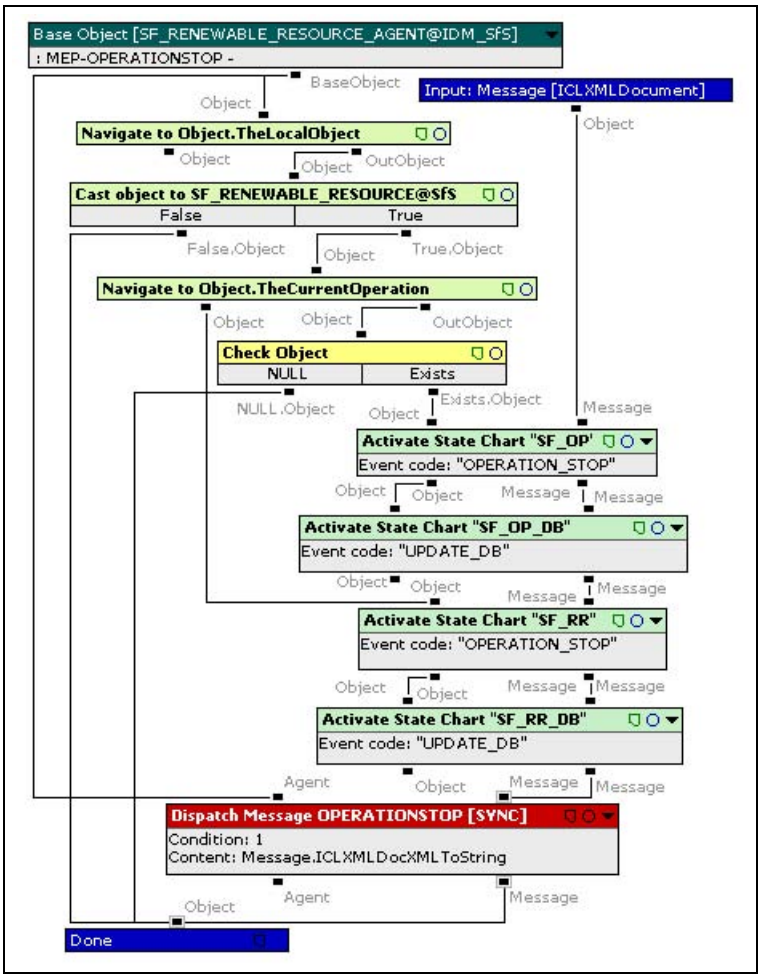

Figure B.3. MEP of SF-RENEWABLE-RESOURCE-AGENT for operation stop message

OPERATIONARRIVAL MEP of SF-RENEWABLE-RESOURCE-AGENT is shown in Figure B.4.

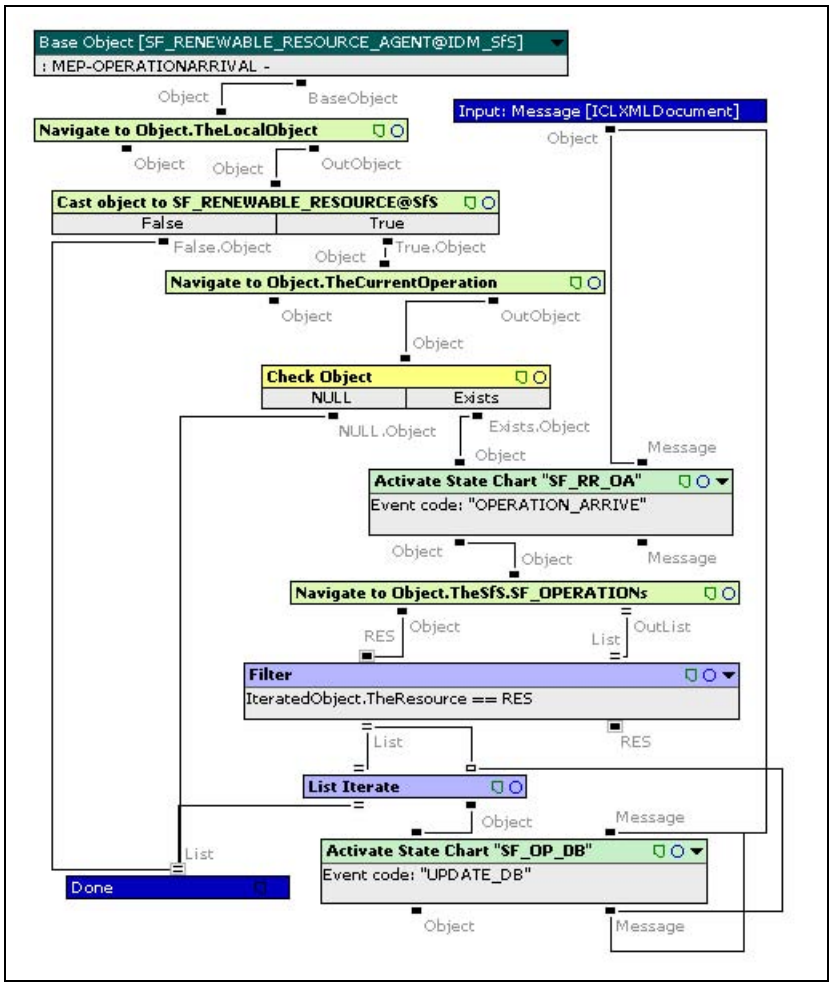

Figure B.4. MEP of SF-RENEWABLE-RESOURCE-AGENT for operation arrival message

REFERENCES

Anussornnitisarn, P. and S. Nof, 2005, "Decentralized control of cooperative and autonomous agents for solving the distributed resource allocation problem," *International Journal of Production Economics*, Vol. 98, pp. 114–128.

Campos, A. and J. Navarro, 2004, "A page-coherent, causally consistent protocol for distributed shared memory," *The Journal of Systems & Software*, Vol. 72, No. 3, pp. 305–319.

Caridi, M. and A. Sianesi, 2000, "Multi-agent systems in production planning and control: An application to the scheduling of mixed-model assembly lines," *International Journal of Production Economics*, Vol. 68, No. 1, pp. 29–42.

Gou, L., P. Luh and Y. Kyoya, 1998, "Holonic manufacturing scheduling: architecture, cooperation mechanism, and implementation," *Computers in Industry*, Vol. 37, No. 3, pp. 213–231.

Jeong, I. and V. Leon, 2005, "A single-machine distributed scheduling methodology using cooperative interaction via coupling agents," *IIE Transactions*, Vol. 37, No. 2, pp. 137–152.

Kutanoglu, E. and S. David Wu, 2006, "Incentive compatible, collaborative production scheduling with simple communication among distributed agents," *International Journal of Production Research*, Vol. 44, No. 3, pp. 421–446.

Kutanoğlu, E. and I. Sabuncuoğlu, 2001, "Experimental investigation of iterative simulation-based scheduling in a dynamic and stochastic job shop," *Journal of manufacturing systems*, Vol. 20, No. 4, pp. 264–279.

Kutanoğlu, E. and S. Wu, 2004, "Improving scheduling robustness via preprocessing and dynamic adaptation," *IIE Transactions*, Vol. 36, No. 11, pp. 1107–1124.

Lawrence, S. and E. Sewell, 1997, "Heuristic, optimal, static, and dynamic schedules when processing times are uncertain," *Journal of Operations Management*, Vol. 15, No. 1, pp. 71–82.

Liu, J. and K. Sycara, 1997, "Coordination of multiple agents for production management," *Annals of Operations Research*, Vol. 75, pp. 235–289.

Luckham, D. and B. Frasca, 1998, "Complex Event Processing in Distributed Systems," *Computer Systems Laboratory Technical Report CSL-TR-98-754. Stanford University, Stanford*.

Mahoney, M., A. Bader, T. Elrad and O. Aldawud, 2004, "Using Aspects to Abstract and Modularize Statecharts," *The 5th Aspect-Oriented Modeling Workshop In Conjunction with UML*.

Oğuz, S., 1996, *Object Oriented Design of a Distributed Scheduling System*, M.S. Thesis, Boğaziçi University.

Rashid, A., A. Moreira, J. Araujo, P. Clements, E. Baniassad and B. Tekinerdogan, 2006, "Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design," `http://www.comp.lancs.ac.uk/computing/users/marash/early-aspects/`.

Sabuncuoglu, I. and O. Kizilisik, 2003, "Reactive scheduling in a dynamic and stochastic FMS environment," *International Journal of Production Research*, Vol. 41, No. 17, pp. 4211–4231.

Shen, H. Q. Y. H., W. and D. Norrie, 2006, "Applications of agent-based systems in intelligent manufacturing: An updated review," *Advanced Engineering Informatics*, Vol. 20, pp. 415–431.

Simsek, A. N., 2000, *Distributed Scheduling with Alternative Process Routes*, M.S. Thesis, Boğaziçi University.

Tanenbaum, A. S. and M. V. Steen, 2001, *Distributed Systems: Principles and Paradigms*, Prentice Hall PTR Upper Saddle River, NJ, USA.

Wang, Y., C. Yin and Y. Zhang, 2003, "A multi-agent and distributed ruler based approach to production scheduling of agile manufacturing systems," *International Journal of Computer Integrated Manufacturing*, Vol. 16, No. 2, pp. 81–92.