IMPLEMENTATION OF DISTRIBUTED WORKFLOW MANAGEMENT SYSTEMS

by

Hande Çıtakoğlu

B.S., Mathematical Engineering, Yıldız Technical University, 2003

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

2006

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my thesis supervisor Ali Tamer Ünal for his invaluable contributions, guidance and endless motivation in this study. Without his support this work would not be possible.

I would like to express my gratitude to Ümit Bilge and Levent Akın for being kind enough to take part in my thesis committee and their valuable contributions.

I would like to express my special thanks to my friends for their interest in this work and for their supporting friendship.

Finally, I am deeply grateful to my brother and my parents for supporting me in all life choices I have made.

ABSTRACT

IMPLEMENTATION OF DISTRIBUTED WORKFLOW MANAGEMENT SYSTEMS

In this work, workflow management systems are investigated and a workflow engine which can be used in the competitive and dynamic business environment for managing distributed workflows is proposed. The currently used workflow management systems' insufficiencies revealed the need for the implementation of distributed workflow management systems. Workflow management systems ease the execution of business processes and decrease the processing time but if WfMSs are run on single servers, the workloads increase and servers become overloaded causing the operating time to be increased. Therefore multi servers are preferred for the management of workflows but the workflows have to be defined in each server separately and the definitions need to be kept updated. As the servers have to be kept updated, synchronization operations have to be scheduled frequently which increases the communication costs. Also these servers have to be using the same applications and same language for communication and execution. Another drawback of the current systems is the load of the data that are transferred between the processing systems. The job is completely transferred to the related workflow agent, causing the workload of the server increase by occupying the network. Transferring the whole job causes security vulnerability and increases the risk of loosing the data. The proposed WF engine solves these shortages. In this work, we distribute the workflow definitions between several workflow engines and send only the required data to related WF engines for processing. Using several different WF engines in the proposed system; the workload on the servers decrease, the systems become more robust and scalable, and their performances increase. Flexibility of the systems increase also since the change of related WF definition is enough for the change of the whole WF. As an additional benefit, reference of the data and state of the job that are sent to the other engines are saved in business and instance data stores in the proposed system, supplying data and state persistency in the system.

ÖZET

DAĞITIK İŞ AKIŞI YÖNETİM SİSTEMİ UYGULAMASI

Bu çalışmada mevcut iş akışı yönetim sistemleri incelenmiş; dinamik, rekabete açık iş ortamlarındaki iş süreçlerini yönetebilecek dağıtık iş akış sistemi sunulmuştur. Kullanılan mevcut iş akışı sistemlerinin bazı konularda yetersiz kalıyor olması dağıtık iş akışı sistemlerinin gerekliliğini ortaya çıkarmıştır. İş akış sistemleri, iş süreçlerinin yürütülmesini kolaylaştırırken operasyon yürütme zamanlarını azaltmaktadır. Ancak sistemler tek bir sunucu üzerinde koşturulduğu takdirde sunucu üzerindeki iş yükü artmakta ve artan bu iş yükünden dolayı, diğer iş akış süreçlerinin yürütülmesi daha fazla zaman almaktadır. Bu durumu ortadan kaldırmak için çok sunuculu sistemlerin kullanımı tercih edilmeye başlanmıştır. Ancak, iş akışları bu sistemlerdeki her sunucuda ayrı ayrı tanımlanmalı ve bu sunucular özdeş tutulmalıdır. Sunucuların güncel tutulması için gerekli olan özdeşleştirilme operasyonu sıklıkla tekrarlanmalıdır. Özdeşleştirme operasyonun sıklıkla tekrarlanıyor olması ise iletişim/ağ maliyetlerini arttırmaktadır. Ayrıca bu sunucular aynı uygulamaları kullanmalı ve aynı dili konuşmalıdır. Mevcut sistemlerin diğer bir kusuru da işlem yapan sunucular arasında gönderilen büyük boyutlu verilerdir. Bir işin tüm verisi ilgili iş akışı sunucusuna gönderilirken, ağ sistemi gönderilen veri ile meşgul edilmekte ve sunucularının yükü artmaktadır. İşin tüm verisinin gönderiliyor olması, güvenlik açıkları oluşmakta ve verinin kaybolma riskini artırmaktadır. Sunulan yeni iş akışı motoru yukarıda bahsedilen kusurları ortadan kaldırmaktadır. Yeni sistemde, iş akışı tanımları birçok iş akışı motorları arasında dağıtılır ve sadece gerekli olan bilgi ilgili iş akışı motoruna gönderilir. Birçok iş akış sunucusu kullanılarak, sunucuların üzerindeki iş yükü azaltılırken, sistemlerin daha olgun ve ölçeklenebilir olması ve performanslarının artması sağlanır. İş akışındaki herhangi bir değişiklik için, akışın ilgili kısmının, ilgili sunucuda değiştirilmesinin yeterli olması, sistemin esnekliğini de artırmaktadır. Sunulan sistemin ek bir katkısı olarak, referansı gönderilen verinin tamamı Görev Veri Tabanında (Business Data Store) ve işin statüsü Statü Veri Tabanında (Instance Data Store) saklanır. Böylece veri ve statü kalıcılığı sistemde sağlanmış olur.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

BDS	Business Data Store
BO	Business Object
DB	Database
EQ	Execution Queue
GSAMS	Graphical Scheduling Algorithm Modeling System
GUID	Globally Unique Identifier
ILP	Input Link Point
IDS	Instance Data Store
OLP	Output Link Point
OMG	Object Management Group
SQL	Structured Query Language
TCP	Transport Control Protocol
XML	Extensible Markup Language
WF	Workflow
WfMS	Workflow Management System
WfMC	Workflow Management Coalition

1. INTRODUCTION

Everyday most of the companies are re-engineering their business processes in order to be more effective and productive. Nowadays, business process management (BPM) systems are preferred instead of manual document interchanges since they minimize the time required to perform a process. In addition to the time lost, data duplication may occur with the use of document interchange method. Electronic data interchange methodology and e-business applications are developed to overcome these problems. In e-business applications, workflows are formed to model business processes and to control the execution of these processes.

While the business processes become more complex, the business environment changes dynamically and expands quickly; the need for different ways of data sharing and workflow management techniques reveal. Companies that start to perform their jobs in several locations started to search for better ways of planning and executing their processes. Using several workflow management systems, the existence of a central management was inevitable. But managing such distributed remote systems with a central server was too difficult and complicated. Hence the use of distributed management systems, which help managing all servers separately with the existence of a single controlling server, emerged.

In distributed workflow management systems, each of the remote systems is managed on their server sides and a general workflow is executed in order to communicate and control them. The workflow defines the execution line of the processes and ensures that the successor task communicates with the finishing task, by carrying the relevant data through them. Messages are sent to the servers that process the successor tasks when a task stops. These messages can be sent over TCP or SOAP protocols or MQ messages can be passed between the processing servers in order to guarantee that the message is received and executed by the listeners of the workflow systems.

In this work our aim is to find an optimum way to manage all kinds of distributed workflows. Today, the most important problems of the rapidly enlarging technology, around the workflow systems are generally the synchronization of the process activation messages, persistence of instances or data, and the synchronization of the processing tasks.

The synchronization of the process activation messages depend on the durability of the servers that send and receive the messages. If they operate successfully all the time, none of the messages are lost and all the processes are executed according to the received messages. In our problem, if a problem occurs at an operating server, the messages are stored in queues, by this way they are not lost but as they are not sent to the waiting listeners at the right time, some delays occur during the process of a task. The only way to avoid this kind of delay is to maintain the servers in good condition.

The second problem of workflow management systems are the persistence of the instances and data. While a workflow is running, according to the sequence of the tasks, the process may stop, like stopping for an approval, and during this time, if a breakdown occurs, the states of the instances and the on hand data are lost. This may end up with two possible cases:

1. When the system is available for processing again, the workflow has to start from the beginning, causing duplication.
2. The system may not be aware of the problem that occurs during run time. When the system is available again, if the workflow is not started again, the stopped task can not be reactivated automatically therefore the stopped workflow ends without completion.

In order to find a solution to these problems, we developed a system that saves the states of the running instances and the on hand data if a stopping case and a breakdown occurs. In this work, when a process stops, a message is sent to the related listening port. This message sends information about the stopping process so that the remote systems can be aware of the stopping instance and start their work for reactivating the stopped process. When the stopped process is ready for working again, it is taken from the list of instances to be reactivated and put in the execution queue. In the execution queue, all the processes are activated on the basis of first come first served rules. When all the processes that are

put in the execution queue before our instance, in order of arrival, are activated, our stopped instance's turn comes and it starts working.

On the other hand, the instance needs its prerequisites to be completed and its input link points to be carrying the input data for running. Hence, our system needs to save the input link point data that are carried on the instance of the stopping node's input link points. When all the inputs of the instance are on hand, the system puts the instance in the execution queue and runs the instance from the point it stopped.

Using the instance and input data saving system, the persistence problem can be solved. None of the tasks to be run can be missed.

The messaging operation between the running process and the listener ports provide the information transfer in the system. Although there may be delays due to the server breakdowns mentioned above, the synchronization problem of the processing tasks can be solved to a degree with the messaging operation. The information of the tasks that are waiting to be processed is sent to the responsible servers and they are executed when they are ready. By this way the synchronization problem is solved.

The rest of the thesis is organized as follows: Basic definitions of workflows are given in Chapter 2.1. The improvement in workflow methodology is given in Chapter 2.2 followed by the detailed explanation of Workflow Management Systems (WfMS) and Distributed Workflow Management Technology (DWfMT) in Chapter 2.3. A survey of existing literature on workflow management systems is given in Chapter 3. In Chapter 4, the proposed workflow engine and the management of distributed processes in this engine are discussed, detailing the implementation developed in ICRON. A simple workflow example representing the receipt of an order, and the management of remote workflow processes related to the order is given in Chapter 5. Finally, Chapter 6 provides our conclusions and future research areas.

2. PROBLEM STATEMENT

2.1. Definition of a Workflow

Any company or even any business entity has its own methodology for its operations. Having policies and procedures, each methodology is used for managing business rules. In accordance with these procedures, each business process can be defined and controlled. Business procedures are modeled and executed with the help of these definitions either by human or machine. Workflow technology is an enabling technology for coordinating the activities widely managed in manufacturing, banking, accounting, brokerage, insurance, healthcare, telecommunications, customer services, engineering, scientific experiments and etc. When a process is executed and finished successfully, it is documented and reported to the next process again either by human or machine. In early times of information technology development, the processes were run, controlled and reported manually. But as human interaction needs time and effort, the flow of these processes are nowadays preferred to be reported by e-business applications using information technology devices.

Each job operation requires several processes to be run at the same moment. Some of the processes may require other processes to be completed before starting. All the predecessor and successor relations are defined according to the business defined rules and procedures. The complete definition of processes can be called as a workflow.

A workflow defines the individual business processes, the order and the conditions under which the processes must be executed, the flow of data between processes, the users responsible for the execution of these processes and the tools used.

Workflow technology facilitates business process specification, reengineering and automation. A workflow is a collection of tasks organized to accomplish some business process. In addition a workflow defines the order of task invocation or conditions under which tasks must be invoked.

According to the Workflow Management Coalition [WfMC,1995], workflow is defined as the *automation* of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules.

Workflow automation is realized through a Workflow Management System (WfMS). According to the Workflow Management Coalition (WfMC), a WfMS is defined as a system that completely defines, manages and executes workflows through the execution of software whose order of execution is driven by a computer representation of the workflow logic.

Workflow management software is a proactive computer system which manages the flow of work among participants, according to a defined procedure consisting of a number of tasks. It co-ordinates user and system participants, together with the appropriate data resources, which may be accessible directly by the system or off-line, to achieve defined objectives by set deadlines. The coordination involves passing tasks from participant to participant in correct sequence, ensuring that all fulfill their required contributions, taking default actions when necessary.

Workflow model is a definition of the tasks, ordering, data, resources and other aspects of the process. Generally workflow models are defined in a workflow modeling tools also known as process definition tools. Each tool has its own language. In this work ICRON technology is used as the modeling tool for workflows.

A workflow instance represents a particular occurrence of the business process, defined by the workflow model. For example in a bank loan application case, a particular approval of a department or a person in the bank is an instance. When a loan application is modeled in the workflow management system, the execution of its approval workflow starts with the loan request instance.

Workflow instances have states that are identified by the running instances and workflow variables. Workflow variables are local and their life span is that of the instance

itself. An instance can be in “running”, “stopped” or “finished” state. In the loan application example, if an approval is obtained from a department, the instance changes its state to “finished”. In this work we give examples of workflow executions and define local variables.

Dependencies between execution tasks can be defined in several ways:

1. Two tasks can be directly connected by an edge meaning that, as soon as the previous task finishes its job, the next one is ready for execution. When a customer sends his/her loan application to the bank, it is checked by the customer relations department. This is an example of directly connected tasks.
2. Connections between tasks may be performed by special routing tasks like forks that start concurrent executions. *Conditional forks* evaluate the result of the predecessor task, and depending on the conditions that are defined on successor tasks, the ones with True conditions activate. *Total forks* activate the successor tasks when the predecessor ends its execution. In loan application example, if the customer relations department approves the application of the customer, the control operations of several departments like credit control department, etc start. This is an example of the conditional forks. If the executions of the control departments’ operations do not depend on the approval coming from the customer relations department, this case would be an example of total forks.
3. Connections between tasks may be performed by special routing tasks like joins that synchronize concurrent executions. *Total joins* activate the successor task when all the predecessor tasks finish their executions. *Iterative joins* activate the successor task every time a predecessor task finishes its job. If the control operations that are executed on several departments end up with approvals, the contracting department starts its processes for giving the loan that is demanded by the customer which shows an example of total joins.

In the loan application example, several serial and parallel processes are scheduled and fired after the initialization of the approval workflow. Firstly the customer relations department control process, forks a series of sequential approval processes. These approval processes do not fire until the information of the applicant is fully detailed in the system. When the customer relations department control process is finished, several managers are

informed about the loan application and evaluation workflows fire. Depending on the hierarchy in the bank, approval processes may work in parallel and succeed if all of the children processes succeed, irrespectively of which one finishes first. In contrast, these approval processes are conditional connectors, which detect situations where the customer is approved to get the loan. If any of these approval conditional connectors end up with a not approved condition, the loan workflow is ended and a message is sent to the customer relations department informing that the loan is not approved. If all the instances end up with approval conditions, the customer succeeds in getting the loan. Then contracting workflows start to run. Each workflow may run on different servers to decrease the load of each department's server. When all workflows finish, the main workflow for the loaning operation finishes.

Several workflows like the above example can be modeled in business environments. As shown in Figure 2.1 workflow technology (WfMS) integrates the critical factors of business environments such as human labor, processes, and infrastructures. While database management systems (DBMS) take the data management functionality out of application programs, WfMS take the process logic out of application logic decreasing the total amount of work to be done manually.

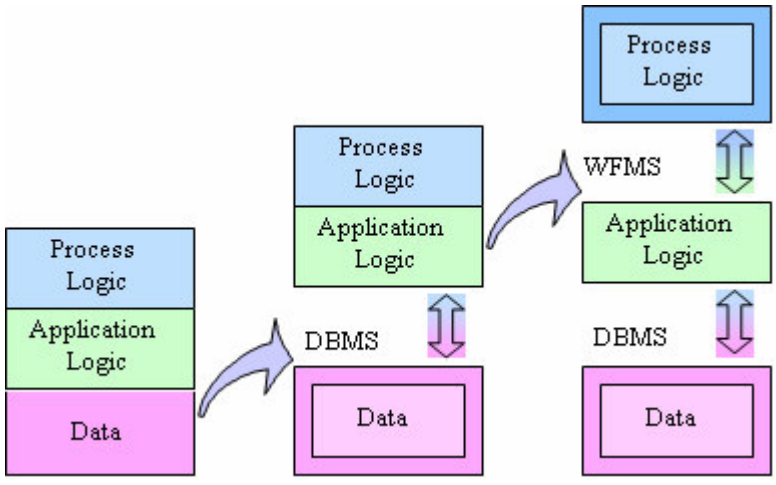

Figure 2.1. Distribution of the Business Logic

2.2. Improvement of the Workflow

As shown in Figure 2.2, workflow technology was used in call centers and offices for managing paper based documentation in the 80s and early 90s. According to WfMC

[WfMC, 1995], human interaction level was very high in those days. Thus failure rate was high causing the success rate of operations to be low. The possibility of paper based documents loss was also high as they were saved by employees in files. Also it was not easy to find the right document at a short time. So as time goes by, with the increasing usage of computers, paper based documents were preferred to be saved as images.

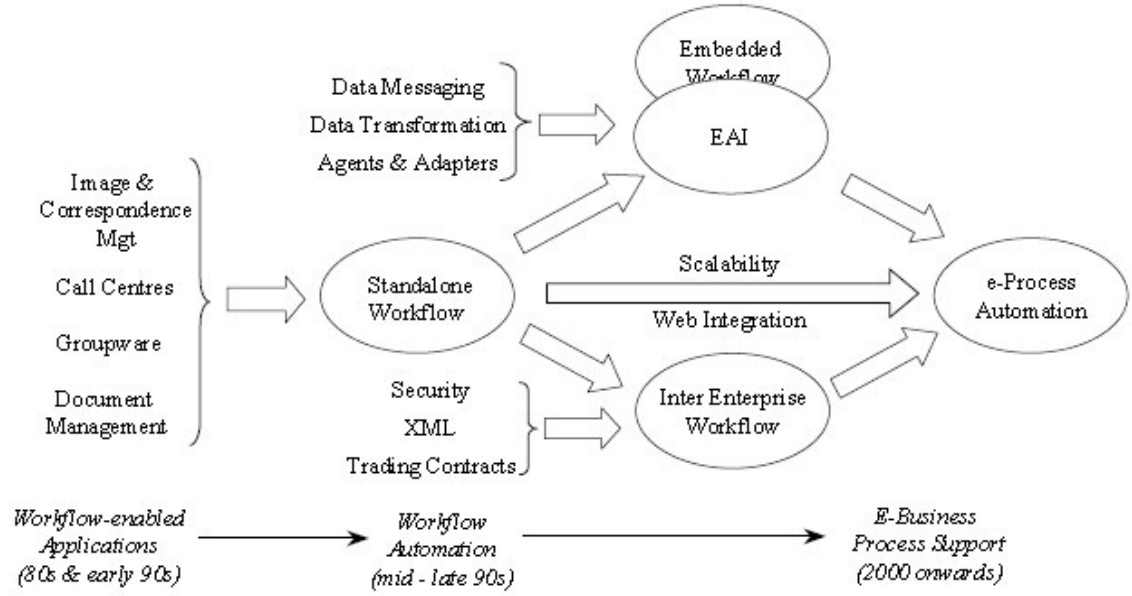

Figure 2.2. Workflow Evolution

Coming to the 90s, it was realized that saving images needed so much space and corruption of the image data caused loss of information. Hence different technologies were found like data messaging. Processes that were needed for sending the messages were handled with organizational workflow automation. All the processes were saved in XML documents.

In contrast synchronization, scalability and performance problems were at hand at the 90s. The breakdown of data messaging servers caused synchronization problems. If the messages were sent while the remote server was not operating successfully, they were lost. And the data sent through messages were carrying the complete definition of the workflow. Again so much space was required for receiving these messages which caused scalability problems. As the data carried on the messages were lost, the workflows ended at any time

revealing unsuccessful completions of operations. Since all the workflows were controlled from single servers, performance problems also occurred.

With the emerging usage of World Wide Web, new technologies called as e-business process management took place in the competitive and rapidly changing market. In e-business applications, workflows are modeled using several information technology tools. The execution of the business processes are controlled with these tools. Today as the business processes become more complex, the need for alternative ways of data sharing and workflow management techniques revealed.

Business entities that perform their jobs in several locations start to distribute their operations and manage them using workflow management systems, with the existence of a central server. Using a single server, there are no synchronization problems but the main problem is the low performance of the server. Some operations lock the memory of the server blocking other operations to run. Hence the use of distributed management systems, which helped managing all servers separately with the existence of a single controlling server, emerged. Each operation runs on its server without knowing the ongoing operations on the other servers. They just send the needed results to the needed servers. Not knowing the other operations and not seeing the data they are carrying also supplies higher security. Control of the communication and the security of these servers is managed by a single controlling server which is the only entity that also knows the order of the operations.

2.3. Workflow Management Systems

According to WfMC [WfMC, 1995], a WfMS is a system that defines, creates and manages the execution of workflows through the use of software, running on one or more workflow engines, which is able to interpret the process definition, interact with workflow participants and, where required, invoke the use of IT tools and applications. Figure 2.3 represents the architecture of WfMSs.

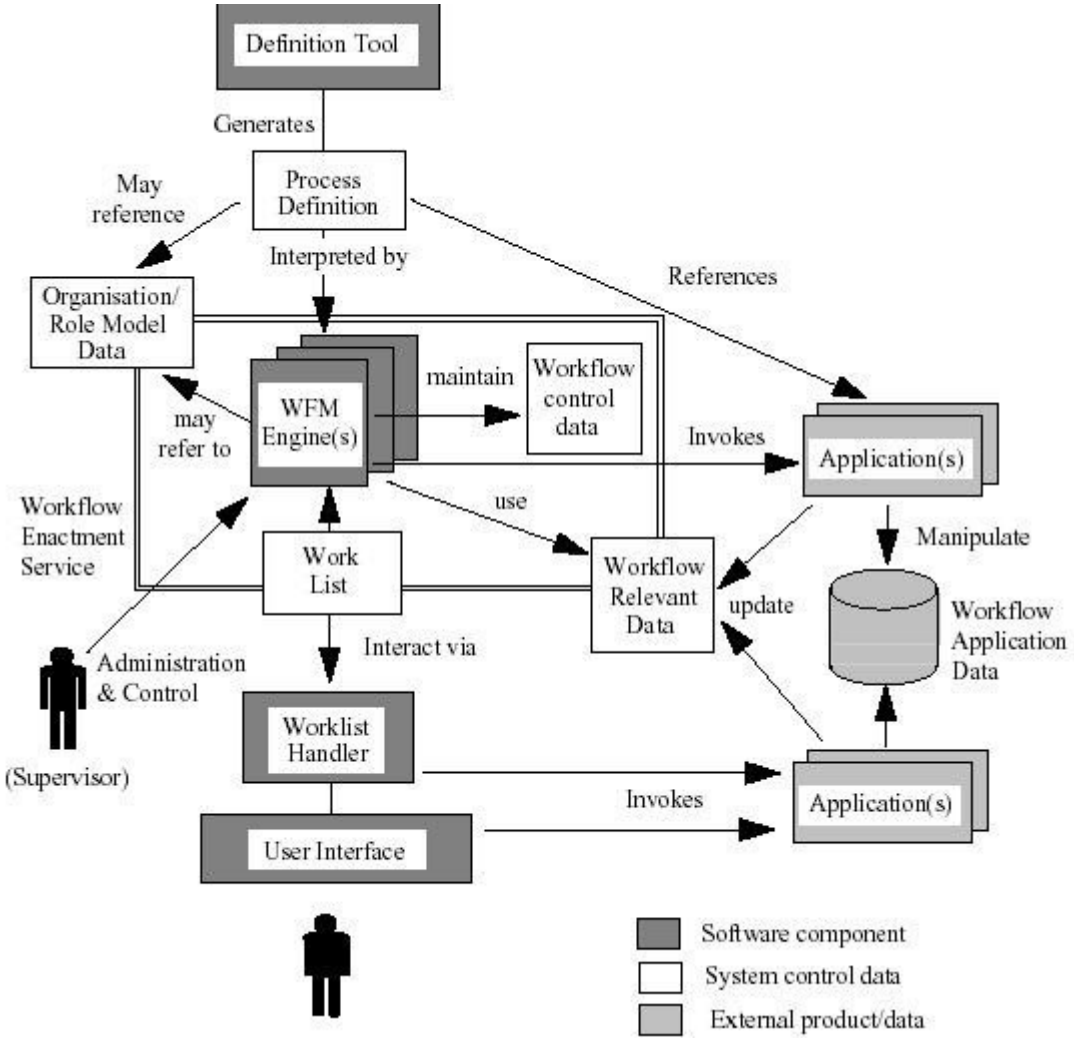

Figure 2.3. Workflow Management System

According to Alonso [Alonso et al., 1997] a WfMS is an active system that manages the flow of business processes performed by multiple persons. It gets the right data to the right people with the right tools at the right time.

At the highest level, all WfMSs may be characterized as providing support in three functional areas:

1. The Build-time functions, concerned with defining, and modeling the workflow process and its constituent activities
2. The Run-time process control functions, concerned with managing the workflow processes in an operational environment and sequencing the various activities to be handled as part of each process
3. The Run-time activity interactions, concerned with human operations while using IT application tools for processing the various activity steps

Figure 2.4 illustrates the basic characteristics of WfMS that are defined by WfMC [WfMC, 1995] and the relationships between these main functions.

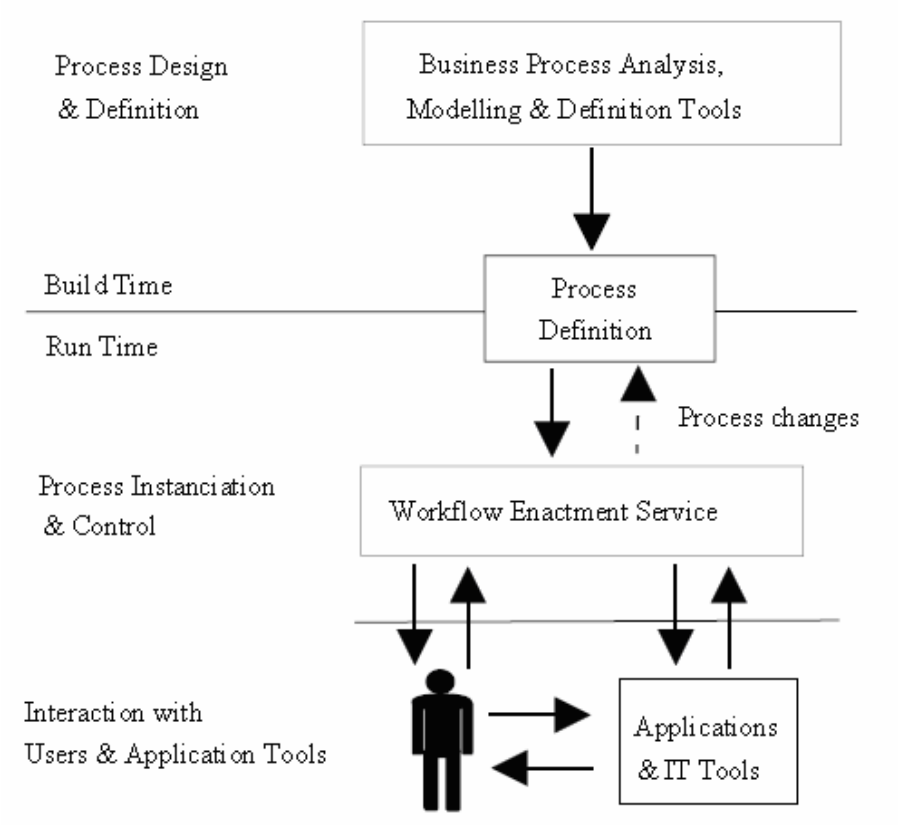

Figure 2.4. Workflow System Characteristics

With the growing acceptance and need of workflows, the technology is used more in large organizations. For wide applications, the load of the WF servers and the amount of

communication may end up with bottlenecks. In these cases, great insufficiency in scalability, performance and reliability occurs.

Our aim is to provide better scalability of processes, clients or instances and better performance by minimizing the load of the servers. This can be provided by the use of workflow management systems processing on distributed servers.

But why are distributed WFMSs used? What are the most important advantages of distributed WFMSs? They are listed as follows:

1. They provide a mechanism to support process modifications allowing for collaborative style of work. The ability of workflow processes to change when the business process evolves represents the dynamism of the workflow systems. The evolution of the business process may be slight as for process improvements, or drastic as for process innovation or process reengineering. In each case, workflows can be adapted dynamically.
2. They consider collaboration between heterogeneous systems at the process level and WFMS level. They are so flexible that they can be customized for every kind of processes.
3. They provide better monitoring of functions and tools to oversee many different views on the process. The instances of a workflow can be easily traced.
4. They consider most applications for web based systems with business process semi-automation.

In contrast with these advantages, WfMSs have some specifications that can be listed as disadvantages:

1. The workflow engine is the core of any WfMS. Although WfMS modules can be installed separately and optionally to remote servers, they have to interact with a specific workflow engine that supports these modules.
2. There are not many techniques available commercially, that allow workflow engines or modules work integrated with other applications. This is a

disadvantage of WfMSs, which forces all modules to work using the same application interface.

3. The workflow process defined in a WfMS may not be accessed and executed in another WfMS. Mostly they are not even transferable. This is another disadvantage of WfMSs that reveal the need for standardization.

Several workflow-related standards have emerged to promote interoperability between different WfMSs. The most significant institutions that have published standards are the Workflow Management Coalition (WfMC) and the Object Management Group (OMG). They specified a reference model describing a set of common interfaces for WfMSs. The interoperability interface intended to interconnect the participating workflow enactment services.

Communication interoperability between WfMSs can be realized by:

1. Direct interaction of WfMS via a set of standardized functions.
2. Message passing between WfMSs
3. Use of shared data stores like commonly accessible data repositories
4. Bridging of systems using gateways to connect different protocols and formats

The first technique requires all the workflows to be running on identical WfMSs that use the same languages, functions and definitions. Today's WfMSs are far from having a common meta-model for business process definitions. Due to the lack of a common meta-model, the resulting workflow definitions are often not re-usable for other WfMSs in other companies. The adaptation of a common process definition language (a meta-model) will allow successful interoperability of different workflow engines. WFMC has developed such a meta-model named as WPDL (Workflow Process Definition Language) for describing the process definitions and also a textual grammar for the interchange of them. Also there is XPDL (XML Process Definition Language) which is used for describing and interchanging XML based workflow process definitions.

The second technique requires all WfMSs have a standardized way for messaging. The content of the messages can be converted to different formats that each system resolves. But the messages that are sent and received must be in the same format. In this

work, we use this technique. We send TCP messages that carry the data in XML documents and convert the documents to message strings at the listener side.

The third technique uses data repositories kept at a given address. A service like SOAP collects the data coming to this repository. When a request is sent to this repository, the result of the request is found using the collected data and it is sent to the request sender through SOAP messages. This technique is also developed in ICRON. If we send SOAP messages instead of TCP messages in our implementation, this communication technique will work.

The last technique uses gateways, that convert different messages having different protocols and formats to the format that the receiving system can understand. These systems need to know the general protocols in order to convert them.

3. LITERATURE SURVEY

In this section, brief explanations about the relevant literature published on workflows and workflow management systems are presented.

We classify the similar works and state the differences between our work and the published papers. Up to now several approaches were tried for maintaining the successful execution of business processes. As workflow systems are designed for facilitating the daily operations of business environments and enterprises, the need for reorganization and improvement of these operations reveal everyday. This trend creates a huge market request for workflow systems which assists many commercial workflow modeling applications to be developed. The structure of the developed applications can be basically classified as follows:

1. Central Workflow Management System (Single Server)
2. Central Workflow Management System (Multi Server)
3. Central Workflow Management System Using Standardized Language
4. Distributed Workflow Management System

Each of these applications has benefits and insufficiencies that are to be explained in the following sections.

3.1. Central Workflow Management System (Single Server)

Some of the early users of workflow systems in small enterprises preferred to use central workflow management systems with a single server for answering questions like what is to be processed, how and by whom it is processed. [Georgakopoulos et al. 1995] Such simple commercial WfMSs developed for office automation can support document management, imaging, application launching, human coordination, collaboration, and co-decision. The most important advantage of using single server is helping several agents communicate using the same language for defining tasks, processes, etc. There is no need

for decoders or encoders. Also according to Alonso, [Alonso et al. 1997] worklists of each processing actor are defined easily in central servers. The management of all worklists is easy to maintain in single servers since each processing actor needs to connect to the main server for retrieving its own worklist. If a change in the definition of a workflow item occurs, update of the related worklist is enough for the reorganization of the whole workflow definition. When two actors, responsible of the same processes connect to the main server for retrieving their worklists, the server chooses the first one that connects for retrieving its task and assigns the job to it. This assignment operation can be easily processed and monitored in central management servers.

Although many of the central WfMSs meet the requirements of the basic processes listed above, they allow only limited interoperability with other processing servers. The communication of all remote processing servers are only obtained with the central server, each server does not communicate with the other server. Hence a problem in the main server causes the failure of the whole system. In addition, a central WfMS that controls all workflow instances using a single server causes bottlenecks. Most of the workflow instances end up with tasks waiting to be processed and the only server becomes overloaded very soon.

Despite the fact that using single server improved communication and supplied easy management, it caused bottlenecks and failures in the whole system that is unacceptable in the production or banking applications. Also the lack of interoperability revealed the need of using several servers for management and control.

3.2. Central Workflow Management System (Multi Server)

In order to solve the bottleneck problem, many approaches suggested using a multi-server WfMS with distributed workflow control. [Muth et al. 1998; Bauer 2001; Li et al. 2003] Since single WF servers have the possibility of break downs and failures of the whole systems, most systems use several servers in order to minimize the impact of failures. In doing so, a different server is chosen for the control of each WF activity. By this way, different processes can be run on different servers and when a failure occurs in a server, just the executions of the processes that are run on the failed server stop.

For some business applications, the failure of a single workflow activity is also unacceptable, so different techniques are used to solve the availability problem of the servers like using mirror databases. [Alonso et al. 1997] By replicating the database of each server synchronously, a mirror system is kept as backup. When the main system fails, mirror system starts to run. If there is a change in the workflow, the mirror system has to be changed also, so keeping the exact replica of the system is difficult and costly. The cost of mirroring can be decreased by keeping the mirror system slightly out-of-date. Even each change can be sent to the mirror system and just stored there without applying them. If the mirror system has to take over the execution process from the failed system, it has to apply all the changes first and then execute the process. This may take some time which is also important for some critical applications. As a result, having several servers operating for the workflow; so much time will be lost while waiting for the updates to be applied or it will be too costly to keep synchronous replicas of the operating servers. Hence processes are categorized as Critical, Important and Normal. For critical processes that must be immediately resumed in case of failures, synchronous backups are kept in mirror systems. For important processes that should be resumed eventually but some delay could be accepted, changes are stored and applied when needed. For normal processes, that may not be needed to be executed for the whole system to go on, no replica is kept and when the system is repaired, it goes on executing the workflow. If all the workflows can be classified like this, this may be a feasible solution. But usually as the workflows are defined with minimum tolerance, mirroring may not be a good solution.

Another favorable application to handle the central management of several servers is to dynamically change server assignments (namely Dynamic Server Transfer) in the WfMS. There are different approaches about Dynamic Server Transfer as listed below:

- During runtime, if a failure occurs, the defined work can be deviated to a server that can operate in the similar way. [Bauer et al. 2004] However, all the client servers related to the processing server should be informed about the change, in order to guarantee the responses to be sent to the new server and sending so much information messages may also block the communication lines between the servers.
- According to Reichert [Reichert and Dadam, 1998] dynamic server changes should not be allowed if the activity is started. In this case a single server can be offered to take over the running the process but choosing this server may cause some delays.
- In another approach, transferring the servers is allowed only when the activities are completed. As the task is completed and the data that should be transferred to the next server is sent, no other client server needs to be informed about the server change at run-time but the client servers have to be informed afterwards in order to supply the other servers to communicate with the new server.
- A last approach allows server transfer only if there is a pre-planned server to take over the remaining operation. This kind of change requires no communication between the main processing servers and the controlling servers as the operations and transferable servers are pre-defined.

There are several ways to schedule workflow processes to executing servers based on defined roles, for example round robin, workload distribution, or manual assignment. [Chang & Jaeckel, 2000] With the first two methods, the task assignments are completely controlled by the workflow management system. These can be done by either server transfer or central management. The last method allows explicit assignment to a specific human worker which is not applicable nowadays.

Using multi server means processing the tasks in several different systems. This approach lowers the number of tasks on a server but this time communication is problematic and costly. Each system using its own language raises the need for encoders and decoders. Also the decoding and encoding processes increase the total process time.

As the systems of different servers do not integrate very well, the processes that are defined in a system can not be used by other systems. Therefore in this approach central workflow management systems are far away from having a common language for business process definitions.

3.3. Central Workflow Management System Using Standardized Language

Having the lack of a common language, several standards which allow true interoperability of business processes supported by different workflow engines have revealed. According to Schulz [Schulz and Milosevic, 2004], who support the standardization approach, Process interchange can be achieved in two ways:

- Commonly used process definition language (meta-model) which allows full interoperability between processing servers
- Interface layer which is a partial solution as it enables interaction at the business process level

The main institutions that have published standardized definitions used in workflow process definition languages are the Workflow Management Coalition [WfMC, 1995] and the Object Management Group [OMG, 1997].

The WfMC is a grouping of companies who have joined together to develop a standardized specifications model named as Reference Model, which includes inputs from all major workflow vendors' WfMSs. A common meta-model for defining the processes, a textual grammar for the interchange of process definitions like WPDL (Workflow Process Definition Language) and WAPIs (Workflow Application Program Interfaces) for the manipulation of process definition data are included in this standard.

3.3.1. Workflow Reference Model

The basic need for standardization has arisen from two major reasons:

- Ongoing support for business reengineering and operational flexibility
- Integration requirements resulting from product customization and market variety

Up to now we have mentioned the rapid changes in the business environment like reorganization of business processes, legislative changes, etc; and explained the need for the flexibility of the WfMSs in order to accommodate these changes. Each individual workflow can be separated into parts and implemented in different business organizations. The customization of products and the distribution of production facilities in the business market require successful integrations between different vendors. In order to supply the success of integration, standardized languages are needed and the Reference Model is developed to remedy this.

The significant parts of the Reference Model can be summarized as the following three categories, each building incrementally on the preceding:

- A common vocabulary for describing the business processes and various aspects of the supporting technologies to facilitate automation. This provided the detailed discussion on how a workflow system could be architected in a general sense.
- A functional description of the necessary key software components in a workflow management system and how they would interact. This was developed in a “technology neutral” manner, to allow the model to be independent of any particular product architecture and implementation technology.
- The definition, in functional [or abstract] terms, of the interface between various key software components that would facilitate exchange of information in a standardized way, thus enabling interoperability between different products. Five such interfaces were identified and became the foundation for the WfMC standardization program.

Figure 3.1 illustrates the components and five interfaces of the workflow reference model.

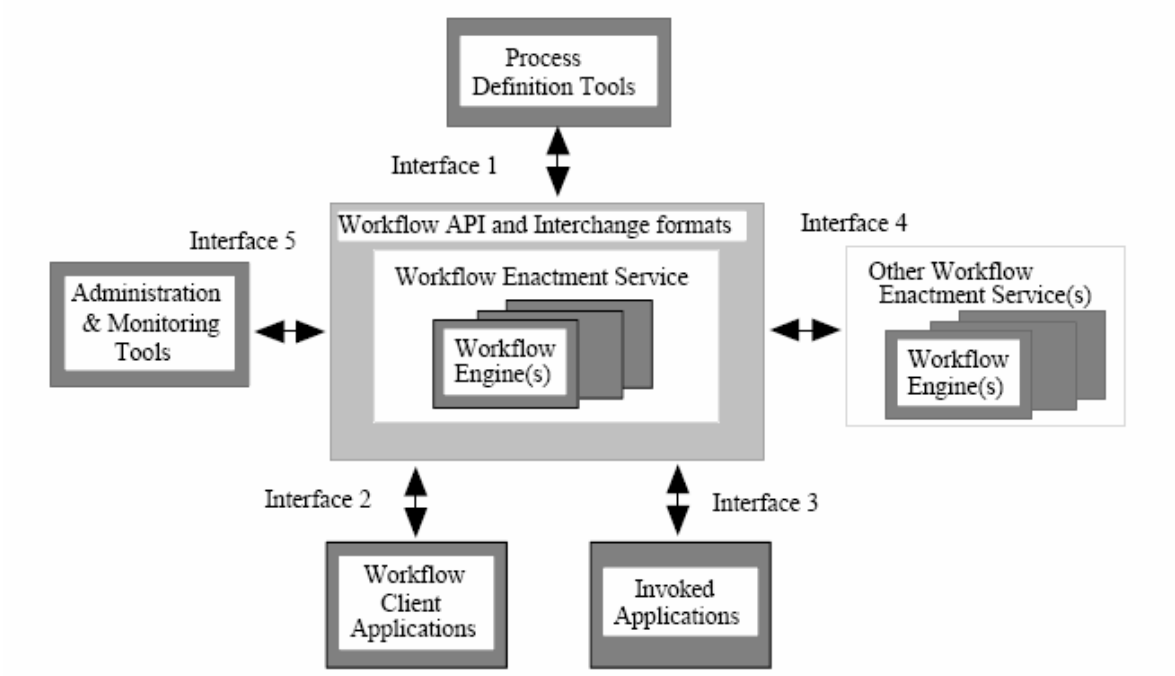

Figure 3.1. Workflow Reference Model (Components and Interfaces)

The Workflow Enactment Service is a software service that consists of several workflow engines which create, manage and execute workflow instances; interpret the process descriptions, control the flow of processes and sequencing of activities by keeping the worklists of each application tools update.

Each workflow product includes its own Process Definition Tool in its production domain, and the definitions may or may not be accessible by other workflow products. Therefore, process definitions need to be interpreted at runtime by workflow engines in the enactment service and an interface, shown as the first one in Figure 3.1, is used to exchange these definitions into the enactment service. The workflow definition interchange interface enables flexibility in this area using standardized format files and API calls. Standardized format files provide separate workflow engines to co-operate and communicate easily without the need of a decoder and encoder. The coalition has developed a meta-model (language) for the process definition which identifies basic set of object types used in the interchange of simple process definitions. This language is called XPDL and it can be expanded by adding object types of several vendors' and conformance

levels with added functionality. Basic Process Definition Meta-model consists workflow process names, version numbers, process start and termination conditions, activities, roles and security, audit or control data. In case of distributed systems, allocations of activities to workflow engines need to be made within the process definition as an additional activity attribute.

The second interface between the enactment service and the user applications hold the worklist containing the queue of processes or tasks assigned to the user side workflow engine. While the worklist is accessible by the user side workflow engine for assigning tasks, it is accessible by the worklist handler defined in the enactment service for retrieving the single task that need to be processed. Through the interchange operation going on at this interface, the worklists are kept updated all the time. In order to supply the updated version of the process descriptions, the standardized API calls and interchanges must contain the process and activity control functions, process status functions, data handling functions and worklist manipulation commands.

The third interface aims to define a standard application interface that allows the workflow engine to invoke a variety of applications through agents. Data interchange depends on the nature of the application invocation interface, it may require the invocation service to embed the data within a specific application protocol and decipher the definitions of the processes running on the agents. The invoked application may run on the local server, it may be co-resident on the same platform or on a separate accessible platform. In each case, the process definitions have to contain the detailed information needed for the invocation operation and they have to be transferable between the workflow engines.

The most important objective of the coalition is to define standards that allow workflow systems used by different vendors to pass work items between each other. Therefore, as the forth interface, definition of workflow interoperability models and corresponding standards to support interoperability is aimed. Both simple task passing operations and complete interchanges of process definitions and workflow relevant data is possible with the use of this forth interface. At runtime WAPI calls are used to transfer workflow relevant data between workflow enactment services. If both enactments services

of the workflow engines support WAPI calls at the same level, the transfer is supplied directly between the workflow engines. In contrast, if the services do not support WAPI calls at the same level, WAPI calls are used to construct a gateway between the two workflow services by mapping different object and data views and supporting different protocols between the two environments.

The last and fifth interface is related to the administration and monitoring operations. Within the context of the reference model, administration operations are run on independent management application servers. They interact with distributed workflow domains in order to control the users, roles, resources; audit the events, change the operational status of workflow processes and their instances.

Corresponding to these five interfaces, conformance is classified at several levels in the reference model. Levels of interoperability between workflow management systems show the level of their conformance. When a system achieves a level of conformance, the system is expected to interact with another system that has the same or lower level of conformance. Some form of interoperability testing or certification that is published by the WfMC is used while specifying the conformance levels of systems.

3.3.2. The Standards Classification

The potential scope of standards that contribute to the BPM lifecycle model is wide. Several overlapping standards exist and hence one of the key problems for process architects and designers is to understand how the various initiatives relate. The diagram, shown as Figure 3.2, was produced by the WfMC Technical Committee. It provides a simple structure for classifying tools in accordance with the functional model described earlier.

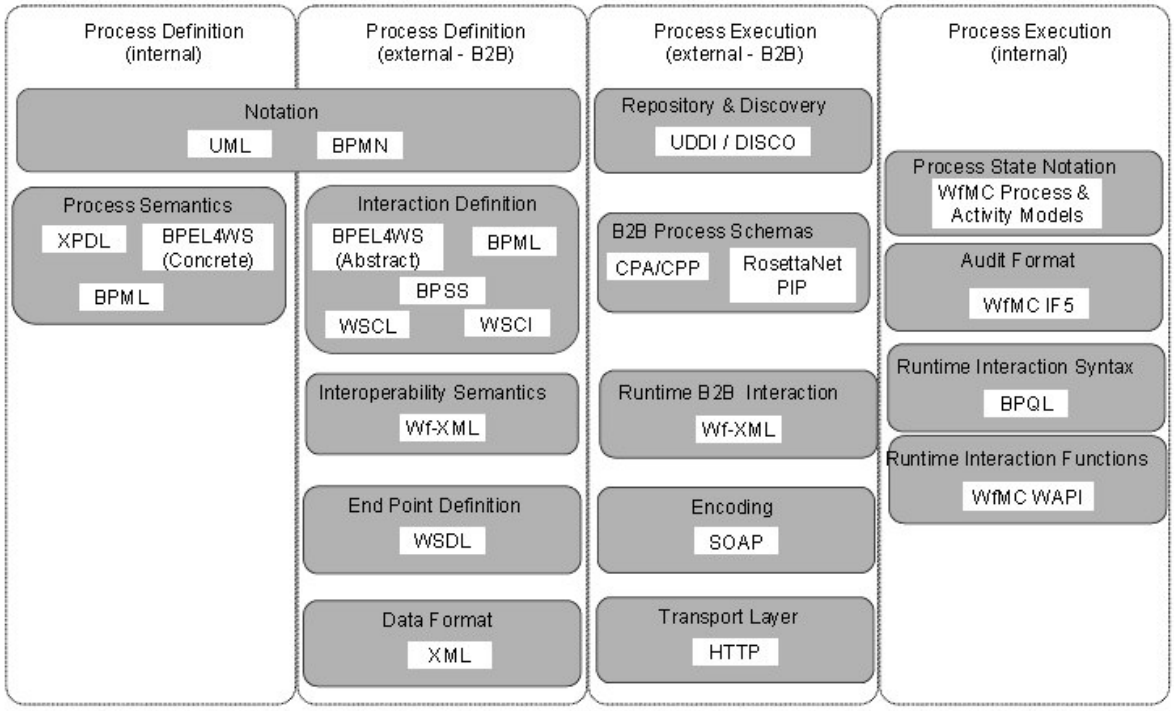

Figure 3.2. Standards Classification

This diagram is based on a standards "stack" aligned with the functional component breakdown from conceptual design tools [at the top] to specific interoperability protocols, formats and encoding [at the bottom]. In the vertical dimension it reflects the progression from abstract design and modeling, through to concrete process and message interactions.

In the horizontal dimension the diagram separates the Process Definition phase (1st & 2nd columns) and Process Execution aspects (3rd & 4th columns). It also separates internal and external ("B2B") views of the process—in both definition and execution phases.

Looking briefly at each column in turn:

a) Internal Process Definition

In column 1 the main components that may need to interact in a standardized way are in the design and modeling domain. So the lower layers of the stack are empty. The use of standards in this space is primarily focused on the integration of different software tools—for example enabling a process definition tool to pass a process definition to an execution

environment. Often software from a single vendor environment will be used within a particular organization or department for both purposes. Not all aspects of internal process behavior will need to be standardized or made visible at external boundaries.

b) External Process Definition

In the external space the essential requirement is interoperability. At definition time this covers specification of the permitted business interactions between different process management systems.

In column 2 the upper layers are concerned with standards to support components for E2E process modeling and choreography. Below that are standards defining the semantics of process interoperability (process operations such as start, stop, query, etc)—this corresponds to the abstract service definition for WfMC interface 4 and the different interoperability models. Below that are the standards (based on web services) defining the service endpoints and data formats supporting interactions.

There is some significant overlap in the standards potentially applicable in this area, particularly in the Choreography area.

c) External Process Execution

Column 3 identifies the standards stack necessary to support process execution, starting with standards for discovery of external interoperability services, progressing through standards for the process schema(s) to support interoperability and then the specific standards to support runtime process interoperability. Currently the only identified protocol to support true process interoperability is Wf-XML.

However, it could be argued that other forms of messaging protocol could be incorporated at this level if they are clearly linked to process semantics. For example, numerous HL7 messages are defined for bilateral interaction in a healthcare environment16 and are associated with specific trigger conditions which cause their

initiation and specific actions [to generate defined outcomes] on their arrival. Many other vertical industries have similar messaging standards.

To the purist this may not be regarded as true process interoperability since there is no process id handle exchanged, to support subsequent process based actions between the systems. This is needed to provide asynchronous connection between the end systems, which then allows querying of the remote process state or actioning dynamic change in the process execution (for example to change context data or state). None the less such messaging will remain widely used in many industries, although simplifications are probably possible using a generic process interoperability protocol carrying industry specific context data payload.

d) Internal Process Execution

The standards applicable in this space (column 4) are principally those that provide a common framework to support execution functionality. Not all BPM systems will support all aspects of these standards but they provide a basic level of functionality for achieving common interpretation.

At the highest level is the model of process and activity state notation, which is used to underpin most execution time activities—including API functions, audit data collection and query state (or any other state based interactions). The notation developed allows local extension of the basic WfMC defined states (open/closed/running /terminated etc).

All other standards make use of these state models (for process, and activity within process). The standards defined cover:

- Audit data collection associated with various state change events, including internal resource assignments
- Process or activity status query
- APIs for consistent access to BPM functions from client applications to query or set process, activity or worklist control data

A subset of this internal information, depending upon the policies of the BPM administrator, may be made visible at the external boundary of the process execution environment. This enables cooperating external systems to have visibility of internal states or audit data in a defined manner. This might typically be through a filter or mapping to a commonly agreed set of generic states.

Over time there has been a major shift in the technology used for realization of the abstract interfaces; most of the original architecture is now expressed in XML and as interfaces to web services.

The correct approach is to recognize what standards are needed where in the architecture, and for what purpose.

3.4. Activity Pre- & Post-conditions

In this approach no explicit transitions between activities are declared. The process is defined as a set of activities each having entry (*pre-*) and exit (*post-*) *conditions*; parallelism is implicit and when pre-conditions are met the activity is initiated, independently of the status of other activities within the process. To provide sequential operation, pre-conditions may relate to the completion of a particular prior activity (and by extension to multiple prior activities, providing an "*and-join*" capability). Post-conditions may be used to control looping within an activity.

3.5. Distributed Workflow Management System

Although central workflow systems are useful for many enterprises, Alonso [Alonso et al. 1997] considers central workflows not scalable and inflexible in terms of integrating with other systems. In addition, Alonso listed the limitations of the existing central workflow systems as being designed for small groups, having the lack of robustness and very limited availability. According to him, the tolerance of failures is minimal as the processes running on central workflows are critical and administratively important. The continuous availability of these critical systems is crucial as a short moment of stopping is

not acceptable in some operations like production or banking processes. As a result; Alonso et al. investigated the available workflow systems and indicated the need for distribution of management, regarding the decentralization of the corporation and decision making authority. The increasing availability of distributed management technology like CORBA, JAVA and internet also simplified the use of distributed management systems. Using distributed systems, the detailed information about all everyday activities are not transferred between processing servers, which helped decreasing the workload on the servers, increasing the availability of the servers and keeping the information about internal processes secure. Despite the fact that distributing workflow management systems is beneficial, managing several systems is also problematic. Maintenance of each system must be obtained, the redundancy of flows and data must be avoided, and the flexibility of tasks and servers must be obtained in failure cases in order to replace the failed server with the operating one without affecting the flow of the process. These are still the lacks of most of the existing systems. Our work overcomes these problems.

It is accepted by Ellis [Ellis et al.1995] that the most important characteristics of workflow systems are the flexibility attributes they provide. In an environment that changes quickly, companies need to refine their processes effectively in order to meet the constraints, new market requirements, opportunities proposed by new technology. The problem of workflow evolution is the modification of the workflow during processing. Assume that a change in the law regulations occurred. All workflows need to be executed according to the new rules. Aborting running executions does not solve the problem since it usually ends up with great loss or redundancy of data and work. As a result the workflows are modified dynamically using standardized languages so as to affect the rest of the running workflows. Today, as the business environment of an enterprise changes and expands quickly, the business process management system needs to be capable of modeling inter-organizational process executions dynamically. The business processes and rules must be easily modeled according to the changing conditions. Hence, distributed workflow engines that communicate with standardized languages which help dynamic changes take place simultaneously are used for processing tasks and solving various problems of business processes.

Meng [Meng et al. 2002] proposes a Dynamic Workflow Model (DWM) that enables the specification of dynamic properties associated with a business process model. The DWM is an extension of the WPDL [presented by WfMC 1999] model by adding event, trigger and rule to WPDL's modeling constructs. By doing so, business events and business rules are integrated with business processes. The enactment of a business process can post events to trigger the processing of business rules, and the processing of business rules may in turn enact business processes. They also separate control information (i.e., Split, Join) from activity definition so that each activity definition is encapsulated and reusable.

According to Aalst [Aalst et al. 2002] modeling a workflow is different from modeling a business process. While business process modeling answers the questions beginning with ''What'' (macro-view at higher level of what need to be accomplished), ''Why'', ''What-else'' and ''What-if''; the workflow modeling focuses on answering questions beginning with ''What'' (micro-view at detailed level of what need to be accomplished), ''How'' and ''Who''. Depending on this assertion, we have tried to model workflows in order to solve business process modeling problems.

4. PROPOSED WORKFLOW ENGINE

In the literature there are several approaches for modeling business processes as detailed in the previous section. However each approach has a different drawback. Central workflow management systems, using single management server, has limited or no interoperability between different systems, they get overloaded so soon, and failure rates are high causing the failure of the whole system. Using multi servers for the management of central WfMSs increase the difficulty in management and also synchronization problems occur in such systems. Since each server, managing the workflow are replicas of each other, a change in one of them requires the change of all the others. Therefore dynamic changes in process definitions reveal the need for a difficult and costly update of each server which represents the inflexibility of the system. Also the process definitions have the possibility to become not accessible by other systems as they are run on different systems and servers. In addition, scalability and availability of the servers are problematic in Central Workflow Management Systems. The performances of the central servers are low, compared with distributed systems. Having such troubles listed above, each business entity starts to use distributed systems.

The advantage of distributed systems is separating the workloads in multi servers named as workflow engines; only the related parts of the process are defined in other processing servers. The whole of the workflow is not defined in each engine like the central systems using multi server, so whenever a change is needed in the workflow definition, just the update of the related part on the related engine is enough. However synchronization and persistence problems may occur in distributed workflow systems.

The insufficient part of distributed systems is that, when a breakdown occurs on a workflow engine, the task that is running on that engine stops and therefore the related part of the workflow stops. Since the result of the task is not sent to the next engine that will process the successor task, the flow of the process and the execution of the job stop. As the workflow is stopped, the replies cannot be delivered on time which reveals synchronization problems. Also the job being processed in the workflow is lost. When the server is available again, the system does not know which job was being processed. In addition, the

state of the running instances and data are lost since they are not saved in a database and data persistence problems are encountered. As a consequence of the existing systems' insufficiencies, the job, the state of the task which was being processed and the data on hand are lost. Therefore when the system restarts, it waits for a reactivation operation to begin its work. As the job is lost, the system may either start from the beginning of the workflow or from a random task in the workflow. If it starts from the beginning, it executes the process again; causing data duplication. If the system starts from a random task, it may pass the stopped task without executing; causing the process to be finished with loss of data. In both cases, workflows end up with inconsistencies.

In order to prevent these, in this work we tried to save each job, its instance state and the data carried at the stopping moment in data stores. By this way we could recreate the job objects that are parts of the workflow item as if they are created at the initialization of the system. After this, the system traces the data stores for finding the exact task which was stopped while processing the job and its input data. Having the job related to this task from the business data store, its instance and input data are obtained easily from the instance data store. When all the input link points of the node are filled with input data, the system reactivates the instance from the point it stopped. In a workflow like loan application, some approvals are needed to be gathered from different departments. Each department might be using different systems for processing their tasks like SAP, Oracle, departmental operation systems, ICRON, etc. When an approval task is received in a department, the main workflow stops and waits for the result of this departmental operation. While stopping the main workflow, its state and the last on hand data are saved in the instance data store and a message is sent to the related engines informing that the process is stopped and it is waiting for the result to be received for restarting. Apart from the workflows, in case a problem occurs in the ICRON system, the loan application job is saved in business data store in order not to loose it. The engine that receives the information message about the stopping instance performs its task and returns the result of the task in another message to the stopped engine. The message is triggered and the result is converted into a language that the receiving engine can process. These messages minimize the delays in the system, and supplies synchronization, ensuring that the engines that receive the messages start their operations immediately.

4.1. Architecture of the Workflow Engine

In distributed systems, workflow definitions are distributed between the processing workflow engines. The proposed workflow engine is responsible for the execution of a single workflow item which is a separated part of the whole workflow definition. The workflow items can be separated into smaller worklists for better operability and management. These worklists contain the definition and the relation of tasks assigned to the responsible workflow engine. Figure 4.1 illustrates the structure of the proposed workflow engine and the relationships between the workflow agents.

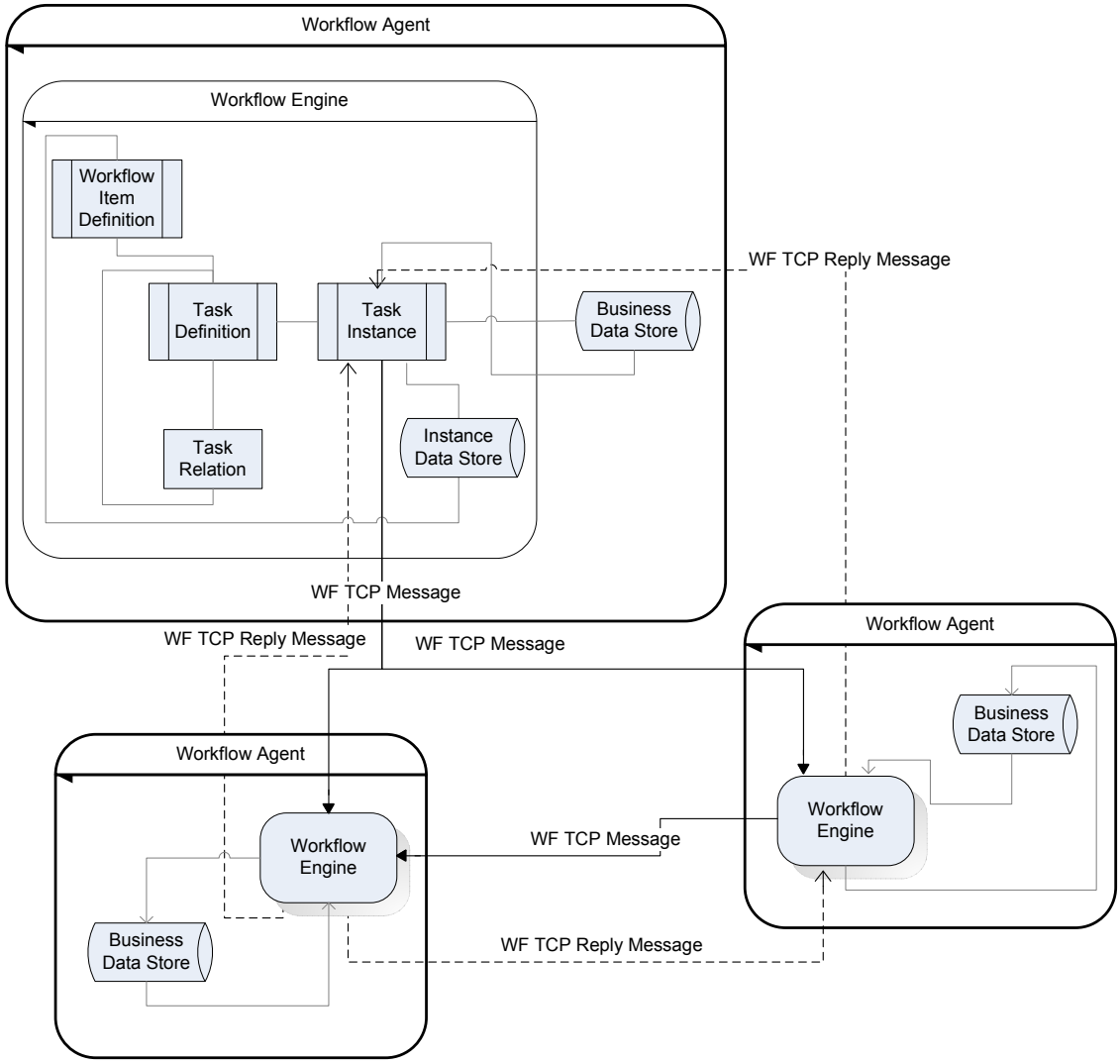

Figure 4.1. Business Logic Managed by Distributed Workflow Engines

The more complex workflows become the better to manage in small sections. Depending on the complexity of the workflow, they can be separated in several systems.

Hence workflows are divided into workflow items and as shown in the figure, workflow items consist of tasks which represent the jobs that are going to be processed within the workflow item. The predecessor and successor relations between tasks are defined in the workflow engine.

While each task is running, an Instance is fired. This instance shows the running task, therefore while the workflow is running, the details of the instance represent the state of the workflow. If all the input data that are needed for the task execution do not reach the workflow engine, the process does not start. In such a case, the process stops and the instance state is set as “Stopped”. Such instances are written into the Instance Data Store in order to supply data persistency and messages are sent via TCP to the related workflow engines that can return the input data to the stopped task. These instances are also registered in the system as Stopped Nodes, so as to direct the return messages which carry the wanted input data, to the correct channel. When return messages are received in the related channel of the workflow engine, the message is parsed and the data it is carrying is transferred to the stopped instance. When all the input data are supplied in the workflow engine, the execution restarts.

Messaging system is improved in the proposed workflow engine in order to register the instance in the channels and send messages to the other workflow engines. Each workflow engine may have its own workflow management system like SAP or ORACLE. These systems can also communicate with each other via messages sent over TCP. They decode the received messages into their languages and process their job. When the job is finished, the output of the task execution is converted to message and sent via TCP to the waiting system.

There is another database in the workflow agent, shown as Business Data Store in Figure 4.1 which holds the business objects, like the jobs, that are related to the workflow item. The jobs that enter the workflow are saved in this database. An example of a job running in the workflow is an order that is entered into the scheduling system. All these jobs are saved in order to be retrieved when needed. In case of failure in the workflow agent, the system may be shut down and if the jobs are not saved in the business data store, they might be lost. As a result, when the system restarts, all the jobs that are not processed

have to be reinserted into the system. Also the references of the jobs can not be found when the stopped instances are retrieved. The instance can not be reactivated without knowing the detail of the job. Therefore, the instances have to be related to the stopped jobs by reading from the DB and reactivated after that.

4.2. Implementation of the Workflow Engine

ICRON is a general purpose visual algorithm modeling tool that is based on Object Oriented Structure [ICRON technologies, 2006]. It has built-in functions that are developed by C++ and defined visually as Nodes in the system. These Nodes are easily used by dragging and dropping on the algorithm pages. Therefore a person who does not have knowledge about programming languages can also use this application by linking Nodes for modeling an algorithm.

The relationships between the Nodes and the algorithm concept in ICRON, resembles workflow structure. While modeling business processes using workflows, the whole of the workflow is separated into work items and separate worklists that contain the related tasks about that part of business process are formed for processing. This structure is similar with the algorithm and node relations in ICRON. Several algorithms can be modeled and related in ICRON for modeling a business process. While the algorithms represent the worklists, nodes in each algorithm represent the tasks in each worklist. Hence ICRON is a suitable tool for implementing workflow management system. Since ICRON can connect with other systems using messages or interfaces, it becomes a more suitable tool for modeling distributed workflow management systems. Within the context of this thesis, we have improved data persistency by saving the processed job data to DB and sending the related part of the data to the interconnected systems. The implementations of these operations are detailed in the following sections.

4.2.1. ICRON Structure

The implementation is modeled on ICRON version 2. ICRON version 1 has been developed as a Graphical Scheduling Algorithm Modeling System named as GSAMS at first. However with the increasing need of the business environment it has been improved for general optimized algorithm modeling. ICRON 2 is a visual algorithm modeling tool which is developed by C++ and works with XML in the base. Several packages (ICRONIA, APS, AgPS, etc.) are implemented within ICRON. In the context of this thesis we have created and used a new package named as WF, representing the workflows.

Being an object oriented system, ICRON has built-in Class definitions. Classes are grouped under Model Classes and Models in general. Models organize the definitions and the relations of the Classes in each package. Users can modify their built-in Classes in their ICRON systems. New Classes can also be defined by users according to their needs. Algorithms that are modeled for running business processes are grouped in these Classes and they are developed by dragging Nodes that are defined in the system with built-in C++ functions. In this work we have also defined some new Nodes in order to use in the algorithms like “Send WF TCP Message” node. By dragging each Node on the Algorithm and relating those with the other Nodes, the flow of algorithms are formed.

While algorithms are processed, each Node creates an instance. The state of the process can be retrieved from the state of the instance. The instance can be in Started, Running, Waiting to be started (Stopped) or Finished state. State of the instance tells us what is going to happen next. When the state of the instance changes into Finished, this shows that the function which is defined in the Node has finished its task and it has revealed an outcome. This outcome is named as the Output Link Point (OLP) Data in ICRON. Nodes also have Input Link Points which carry the data that is coming from the previous node. The OLP Data is transferred to the next processing Node, with the help of Output Link Points. The transferred data becomes an Input Link Point (ILP) Data for the Node that is waiting to activate. Instances have Execution Queues that carry the tasks to be processed one by one. In ICRON, tasks enter these Execution Queues when all the ILP Data they need for execution are obtained. This means that when the predecessor Node does not finish its task and reveal OLP Data, the successor Node waits for the ILP Data to

be received. ILP Data can be either lists of data to be processed or the data itself. When all the ILP Data are set to the Input Link Points of the Nodes, the Node is transferred to the EQ. The first Nodes that are to be processed in the execution queue are the ones that enter the queue first. When the turn comes to the Node in the EQ, it is fired. When all the Nodes finish their tasks, the algorithm ends. Figure 4.2 represents the details of ICRON structure; the relationships explained above between the models, algorithms and instances. Some attributes that are needed for workflow management are also included in the figure.

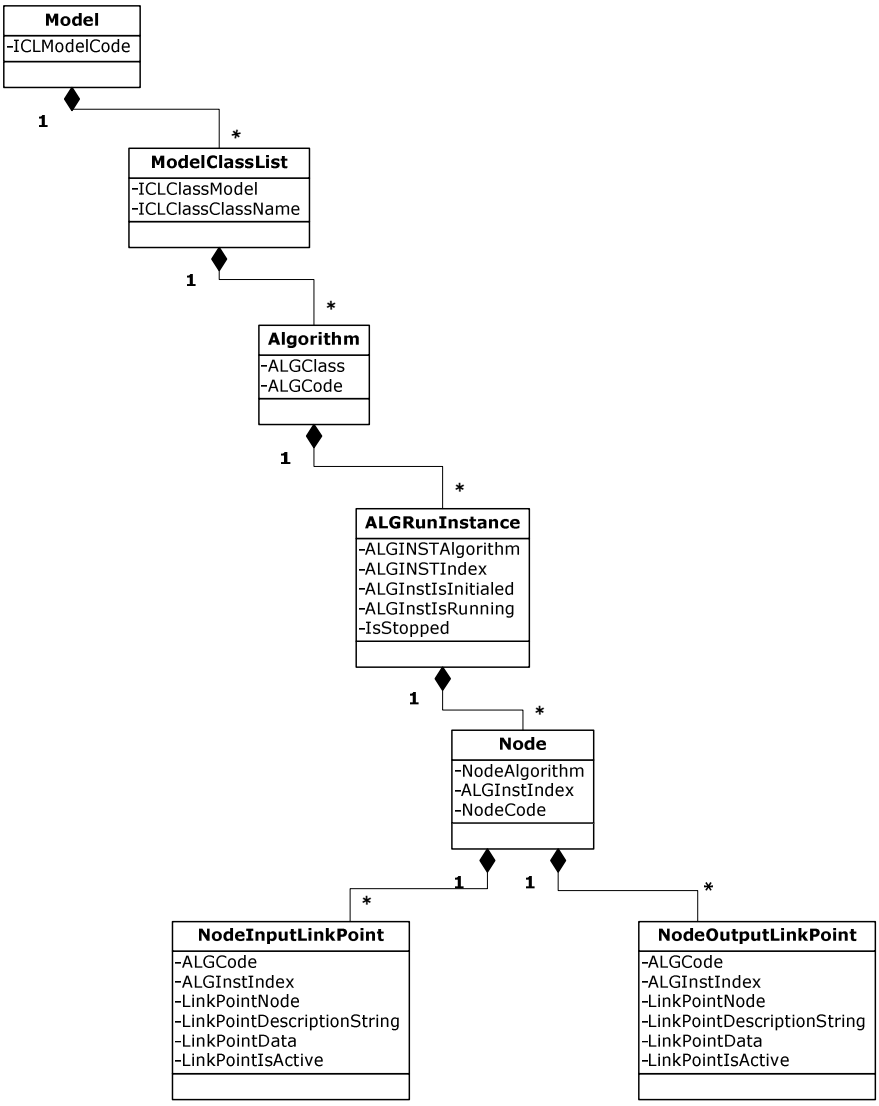

Figure 4.2. ICRON Structure

4.2.2. Workflow Implementation In ICRON

The reasons why ICRON is preferred for implementing the distributed workflow management system is explained in Chapter 4.2. The workflow engine concept is detailed in the motivation part of Chapter 4. The insufficiencies of the existing distributed systems and some new developments which are implemented in order to eliminate these problems are also listed in Chapter 4. There are two major problems in the implementation of distributed WfMSs in ICRON. One of them is the persistence of the jobs. The second one is the persistence of instance states, input data and synchronization. The new implementations that are detailed below are included in ICRON system within the context of this thesis in order to solve these problems.

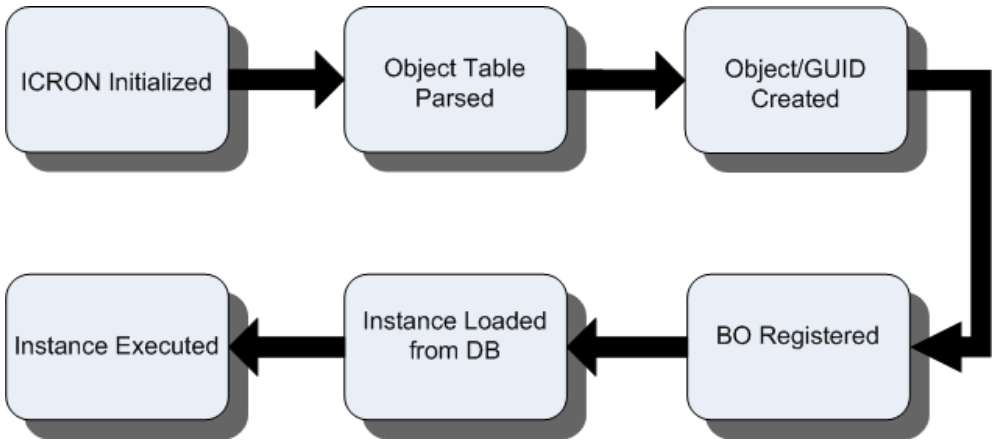

Figure 4.3. Processes that are executed when ICRON is initialized after a breakdown

Figure 4.3 shows the operations that are run when a failure in the system occurs and the server is restarted. Every job inserted in the workflow system is written into the Business Data Store in each workflow agent for forming a list of the jobs processed in the system. In case of failure in the agent, the system is shut down and all the tasks running at that moment are lost. When the server is restarted and ICRON system is initialized, the business data that are to be processed are not available, they have to be reinserted or read from the data store for processing. Therefore, the object tables are parsed in the Business Data Store (BDS) and all the Business Objects (BO) that are going to be processed in ICRON system are created. In this work, we deal only with workflows; therefore we read the JOB table and create Job Objects. If ICRON is used for a distributed scheduling

algorithm, then the SCHEDULE table should be read and schedule objects should be created. These modifications can be easily made in ICRON.

The globally unique identifiers of the new Job Objects must also be created in order to define the Job Objects separately in the system. This operation is important because when the instance references saved in the Instance Data Store (IDS) are compared with the ones that are created from the BDS, these unique IDs, namely GUIDs, have to be matched for choosing the right job and instance to be reactivated. The GUID of each Business Object (BO) can be chosen among the attributes of the Job Object, ensuring that the identifier is unique, like the Code of the Job. In this work we have chosen the GUID of the BO as its Job Code.

After the Business Objects are created; they have to be registered in the system for retrieving when needed. When the system reads the instance table in IDS and finds the stopped ones; it also gets their Job Codes that are saved at the 6^{th} column of the Instance table, as the reference of the instances. In order to reactivate the stopped instances, their sources have to be found. The registered BO GUIDs are compared with the References saved in the Instance table; if a matching job code is found, then the job object of instance that is going to be reactivated is obtained. Using the Instance reference, the Input Link Point Data of the stopped instance is retrieved from the IDS also. Having the Job Object and the Input Link Point Data on hand, the stopped instance's node can be found and loaded in the workflow algorithm. Since all the Input Link Points are loaded, the node becomes ready for execution. It is transferred to the execution queue of the ICRON system and executed when its turn comes.

In this operation, the importance of the business objects is high. Without the Job Object, the source of the instance can not be found and therefore it can not be executed as an Input Link Point Object is empty. As a result, an algorithm that runs the steps explained above is preferred to be executed automatically at the initialization stage of ICRON. As the initialization algorithm runs at each system start up; the BO creation process, and all the steps listed above are processed in order to load the stopped jobs and execute them beginning from the points they stopped.

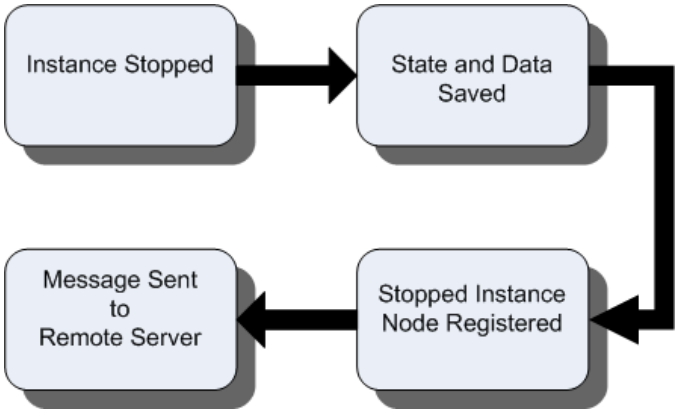

Figure 4.4. Processes that are executed when an Instance Stops

Figure 4.4 illustrates the operations that are run when a process stops at a point. As a new implementation in ICRON, when an instance stops, its state and the data it is carrying at that moment is saved in a database as shown above.

If all the input link points of a node in a workflow cannot be loaded with input link point objects, then this node can not be run in ICRON system. When the input link point objects that are needed for the execution of the node are not obtained, the process stops, the instance state is set as “Stopped” and the instance node is written into the Instance Database as a stopped node. If there is any data on the Input Link Points of the node, they are written into the Input Data table in the Instance DB.

In this work, the database tables are created with Access for saving the instances and data of a workflow that is running in IDS and the job in BDS. Figure 4.2 also represents the design of the database tables created for this work. Our tables save the data in accordance with the attributes stated in that figure.

After the saving operation, the stopped node is registered in the Stopped Nodes channel of the workflow engine. This operation is important because this registration shows that the node is stopped and it is waiting for a message for reactivation. Registration also enables the messages coming from other workflow engines to be directed to the related channel in order to activate the related algorithm with the sent data. After the node is registered; messages are sent via TCP to the related remote workflow engines that can

return the input link point data to the stopped node for stating that the node is stopped and waiting for the result of the remote process's execution.

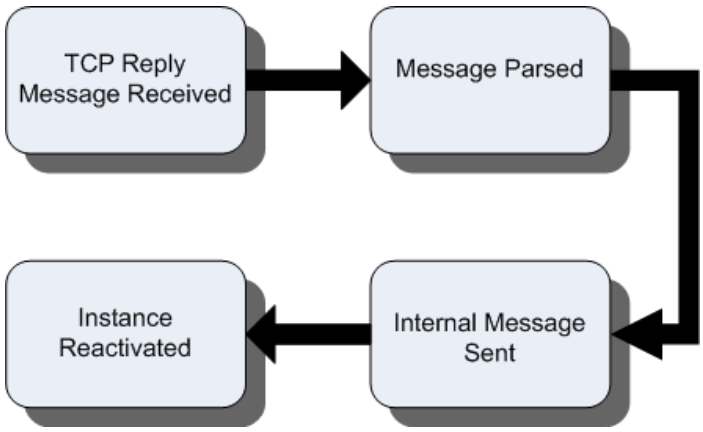

Figure 4.5. Processes that are executed when Reply Message received for reactivation

When the reply message is received at the specified port of the workflow engine, it is parsed and the reference of the message sending server and the data related with the workflow are separated as shown in Figure 4.5. When the data is obtained, it is sent via an internal message to the registered channel for running the processes that will reactivate the related node. When the message is received in the channel, NodeCancelStop function activates and this function changes the state of the node instance into "Not Stopped". Since the state of the node instance is "Not Stopped" and all the input link point objects are loaded with the incoming message, the instance is reactivated. The technical implementations of these operations are detailed in the following sections.

4.2.3. Initializing ICRON System

Within the context of this thesis, ICRON acts as a workflow engine and the server running ICRON application, acts as a workflow agent. The workflow agent contains the workflow engine and the business data store. The BDS holds the business objects that are inserted in the ICRON system. These business objects can be job objects, schedule objects, mathematical model objects, etc. depending on the content of the algorithms. Since we deal with workflows in this work, we will use job objects which. As explained in the architecture of the workflow engine section, business objects are saved in order not to loose the operating objects in ICRON system. If a problem occurs in the workflow agent, the workflow system is shut down and all the operating objects are lost. In order to continue the operation when the system is available, these objects must be obtained. Therefore, the database which holds the BOs is triggered; the saved objects are recreated and registered in the system. Figure 4.6 illustrates the sequence of tasks that are processed while creating new business objects which are used for the reactivation of the stopped jobs.

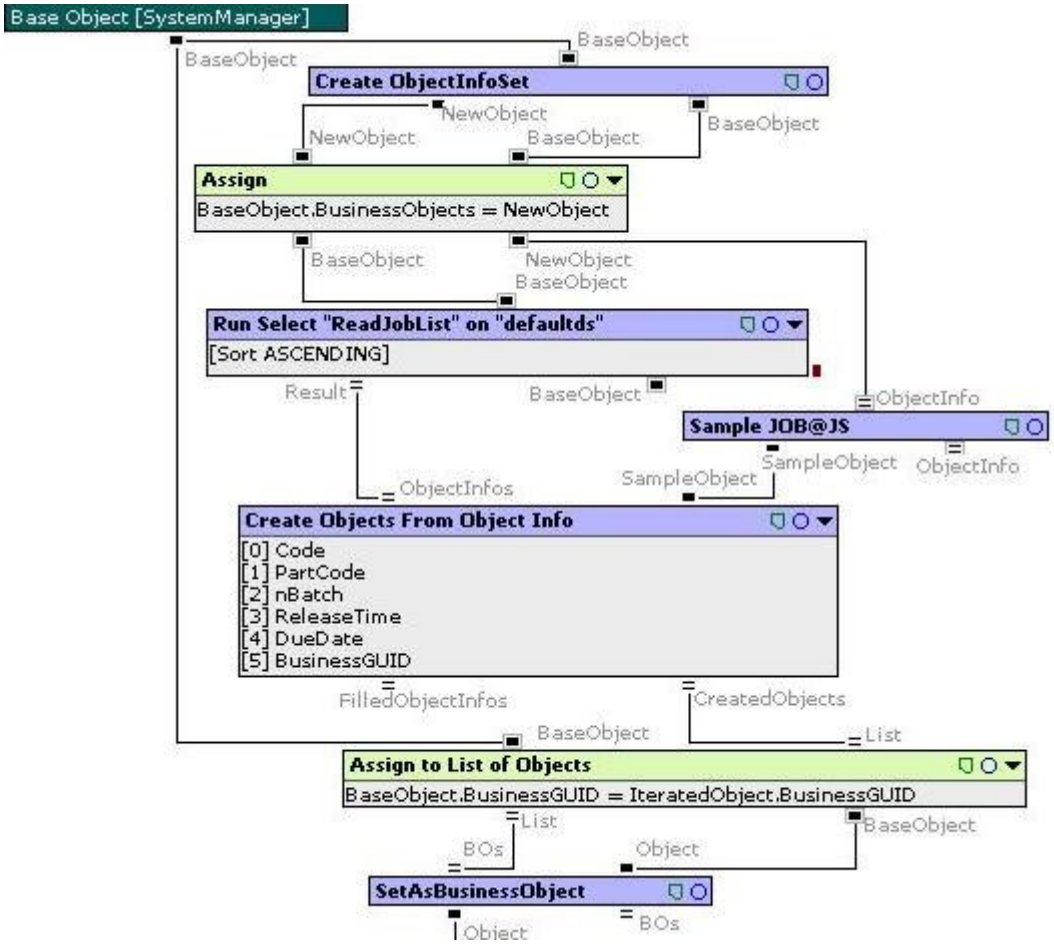

Figure 4.6. Initialization Algorithm

This algorithm runs at the initialization of the ICRON system and creates objects in order to set the references of jobs that are retrieved from the BDS. In the algorithm, the first “Create Object Info Set” node creates a new object and this object is assigned as the business object of the system manager via the “Assign” node. “Run Select” node reads the JOB table on BDS with the help of the SQL statement written in the ReadJobList XML. The created business object is formed as a Job Object via “Sample Job@JS” node, which is a built-in node of ICRON and the job object info read from the table. “Create Objects from Object Info” node matches the info of the job object that are taken from the Sample Job node and the attributes saved in BDS like Job Code, Part Code, Batch Size, Job Release Time, Job Due Date and Business GUID which is set as the Job Code in the context of this thesis. After the job object is created, the Business GUID of the system manager of ICRON system is set as the Business GUID of the created Job Object via Assign to List of Objects” node. Hence the same objects which were on hand before the system was shut down are created again. Finally, “Set as Business Object” node, which is a new algorithm that is written in the context of this work, registers the business objects that have a Business GUID in the system manager of the nnnICRON system.

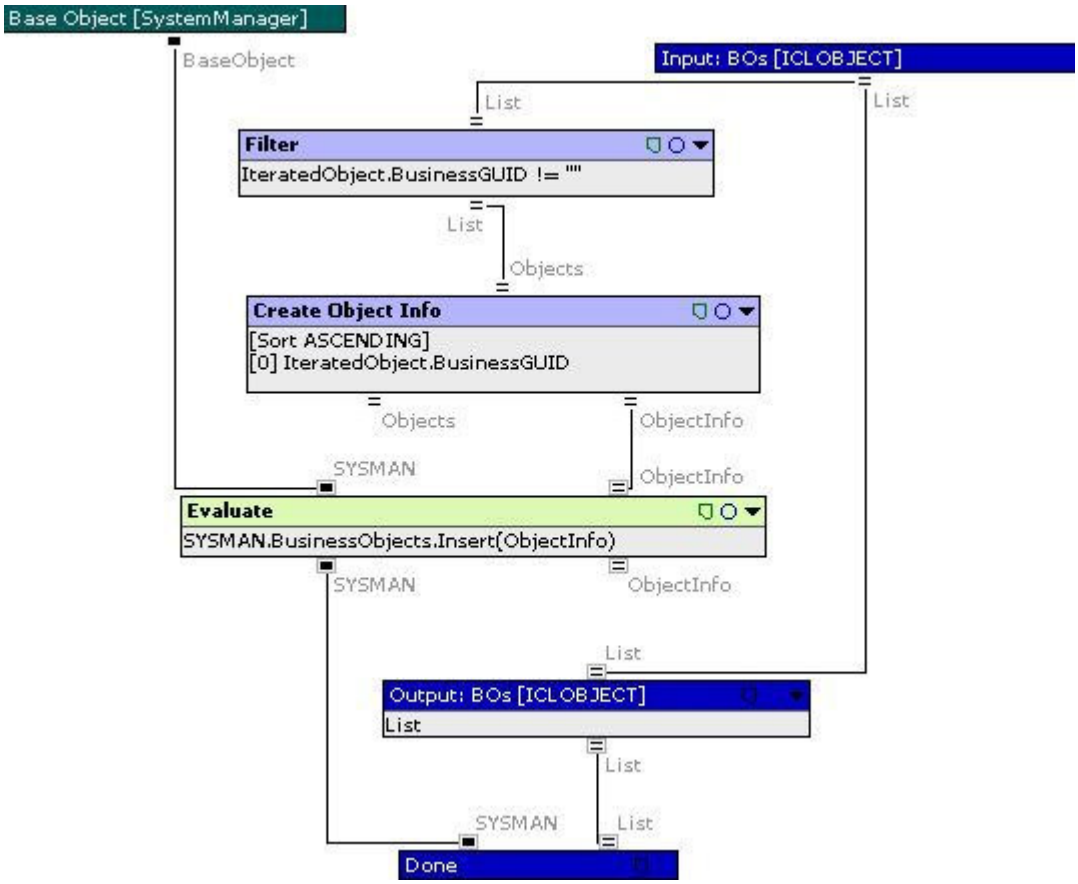

Figure 4.7. Set As Business Object Algorithm

There is another operation that has to be run at the initialization stage. This operation is registering the listeners of the workflow engine. The messages that are sent to the workflow engine have to be directed to specific ports for firing the related algorithms. Therefore listener ports are opened in ICRON system and they are registered in the system specifying that they are waiting for the incoming messages.

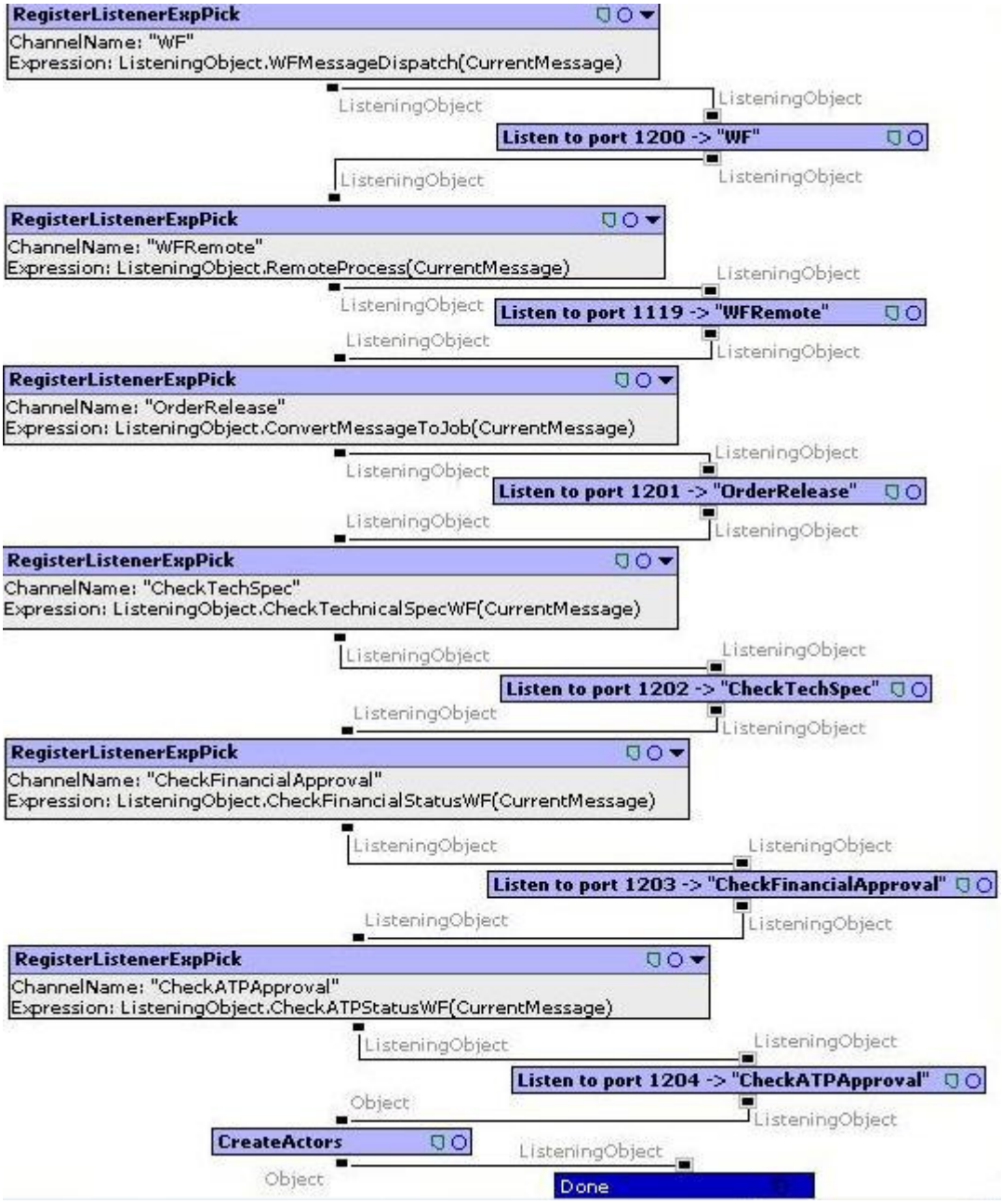

Figure 4.8. Listener Registration Algorithm

In the algorithm shown in Figure 4.8, the first “Register Listener” node registers the listener on the workflow channel. The messages that come to the channel named WF are

executed via the registered algorithm WFMessageDispatch. At the initialization stage, the listener starts to listen to the 1200 port and transfer the incoming messages to the WF channel for executing the message dispatch algorithm. The second register listener node registers the listener on the remote workflow channel. The messages that come to the remote channel named WFRemote are executed via the registered algorithm WFRemoteProcess. At the initialization stage, the listener starts to listen to the 1119 port and transfer the messages to the WFRemote channel for executing the remote process. The confirmation messages of the listeners being registered can be seen at the Threads tab of the Administration explorer.

Each workflow engine registers its listeners in its system. The Ports and the IPs of the workflow engines are saved at the ACTOR table in the DB. Figure 4.9 illustrates the operations that are run for creating the ACTOR objects that are needed for choosing the right server and the right port to send the reply message. In the Listener Registration Algorithm, the last 4 "Register Listener" nodes are used in the workflow management system example detailed in Section 5.

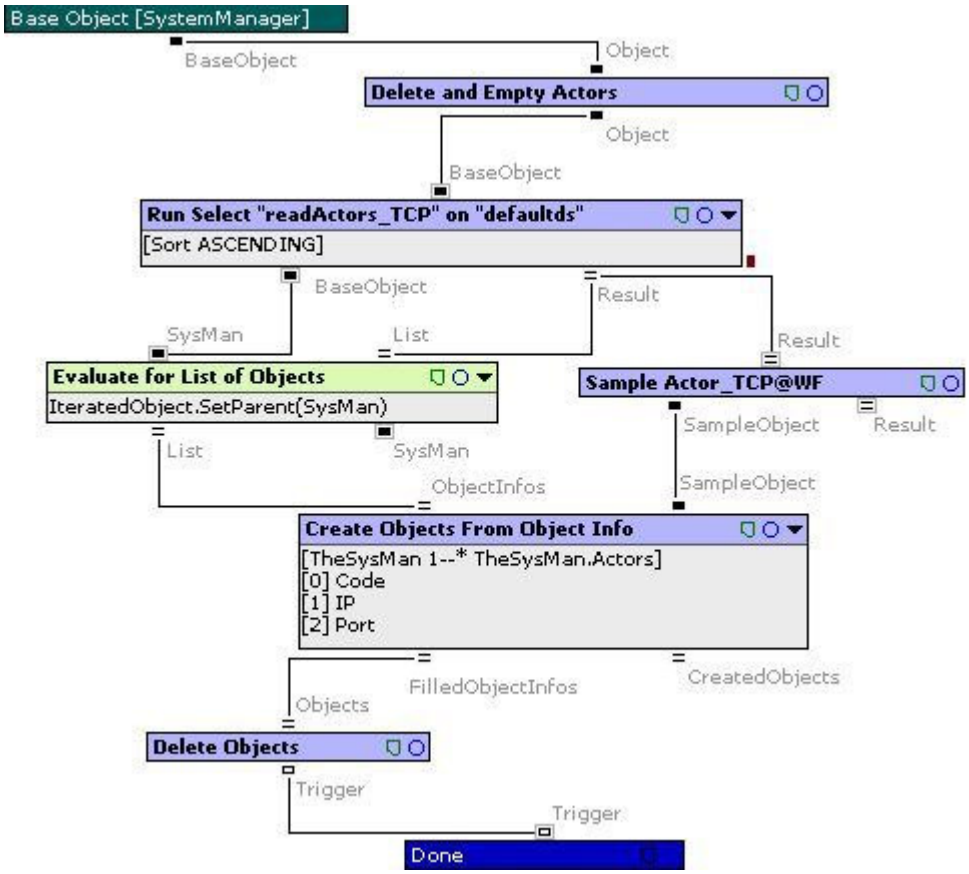

Figure 4.9. Create Actor Algorithm

4.2.4. Stopping an Instance

In ICRON, when an algorithm gets the input data it requires to process, the execution starts. While a task is running, if an error occurs, the execution of the task stops. Another case is that while a workflow is processing, some tasks require other tasks to be done. For example, the confirmation process of a loan application waits for the approval messages of the managers. At this point, the workflow is stopped and the node that is going to be processed starts to wait for these approval messages. When the message that the node is waiting for is received, the node changes its state from “waiting” to “ready to process”.

Now we will explain the functions in ICRON that are used for stopping an instance.

In an algorithm, the state of the instance is checked with the “IsInitialized”, “IsRunning”, “IsStopped” attributes. They are added to the nodes for managing workflow systems with this work.

Special nodes are prepared for the workflow management system. One of them is the “Send WF TCP Message” node. If the workflow requires another task to be processed, it stops and activates the “Send WF TCP Message” node.

Figure 4.10 represents the expression from the Send WF TCP Message node. The functionality of this expression is explained as follows:

```
<Parameter Code="Run_Phase_1">
<Expression> NODEStop </Expression>
</Parameter>

<Parameter Code="OnSTOP">
<Expression>
SendMessageToSocketServer(host, port, NODESTOPTicketID + ":" + msg)
</Expression>
</Parameter>
```

Figure 4.10. Send WF TCP Message Node Expression

In ICRON when this node activates, it runs its first phase which is represented in the Parameter expression of Run_Phase_1. This parameter fires the NODEStop expression that calls StopInInstance function which is defined among the Node functions of ICRON. StopInInstance function stops the running instance, gets the Run Instance Data carried on the processing node and sets the value of IsStopped attribute as “True”.

After stopping the instance, the system checks the unique ID that holds the information about the operating server, such as the IP and the unique code of the server. This ID is defined as StopTicketID in the ICRON system. When an instance is stopped, a StopTicketID that is used for querying the server which was processing the stopped instance is created. While the algorithm runs, StopTicketIDs are not created, as they are not stopped at any time. Hence if the instance is stopped for the first time, it does not have a StopTicketID and when the StopInInstance function runs, having the run instance data, the system creates a GUID representing the StopTicketID. But if the instance is stopped, it has a StopTicketID that can be retrieved.

The next operation that StopInInstance function processes is saving the state of the instance and the StopTicketID of the server. The unique index of the run instance and the created StopTicketID is written to the INSTANCE table having the IsStopped value True. By this way, when the reactivation message is received indicating that the system is ready for working; the index of the stopped instance and the server info that was running the stopped algorithm can be parsed from the table and the algorithm waiting to be started at this server is activated. The INSTANCE table can be checked for stopped instances and one by one all the instances can be moved to the execution queue of each ICRON server for reactivation. While saving the stopped instances in the INSTANCE table, the ICRON Model in which stopped instance is run, the Algorithm in which nodes and instances are defined, the Execution Queue in which the instance is hold and the Input Data that the Instance is carrying at the stopping moment are also saved. These saving algorithms are detailed in Chapter 4.2.7.

After saving the stopped instances in the database, the server registers itself to the STOPPEDNODES channel of the ICRON Listener using the stopped node’s StopTicketID

and the Instance Index Data. ICRON Listener is generally used for getting the messages that come from all the channels to a specific port and channel in the ICRON system. As explained in Chapter 4.2.3, registering a node to the listener means telling the server to get all the messages coming to the specified port and transfer them to the registered channel in order to activate the algorithm registered in the channel which will execute the received message. These ports are saved at the ACTOR table in the workflow engine that helps each operating servers find the IP and the port of the server that they are going to send the reactivation messages. If the instance is stopped and saved at the INSTANCE table, no new StopTicketIDs are created; the node registers itself to the channel with the existing StopTicketID.

When the StopTicketIDs are created and the Stopped node is registered in the channel of the workflow engine; messages are sent via TCP to the related remote workflow engines that can return the input link point data to the stopped node for stating that the node is stopped and waiting for the result of the remote process's execution. The node stays in "Stooped" state until the result message is received in the workflow engine.

4.2.5. Executing the Remote Process (Sending Message for Reactivation)

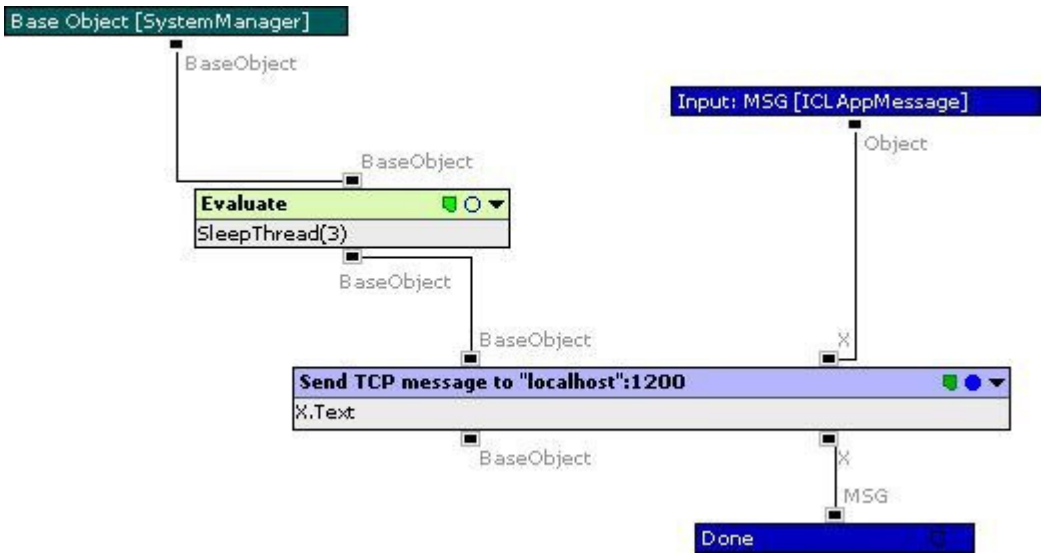

Figure 4.11. Executing the Remote Process

When an Instance stops while running, the operations detailed in the previous section are processed and a message is sent via TCP to the Remote Workflow Engine that will process the wanted part of the workflow item. Remote Process algorithm shown in Figure 4.11 gets the message which indicates that a task is stopped and it is waiting for the result of the sent job. The details of the remote process are not shown in the algorithm above but the important part is that; the remote process executes the process in its workflow engine and returns the result of the execution with a message sent over TCP to the specified port of the workflow engine which holds the Stopped Instance. In this algorithm, 3 seconds represents the time that passes while the system executes the process. After a time given as 3 seconds in the "Evaluate" node, "Send TCP message to localhost:1200" node activates. This node sends the message to the 1200 port specified as the port of the WF engine that holds the Stopped Instance and localhost refers to the IP of the workflow engine. When the reply message is received in the Stopped Instance's WF engine, the operations that are detailed in the following chapter are processed.

4.2.6. Executing the WF Message Dispatch Algorithm

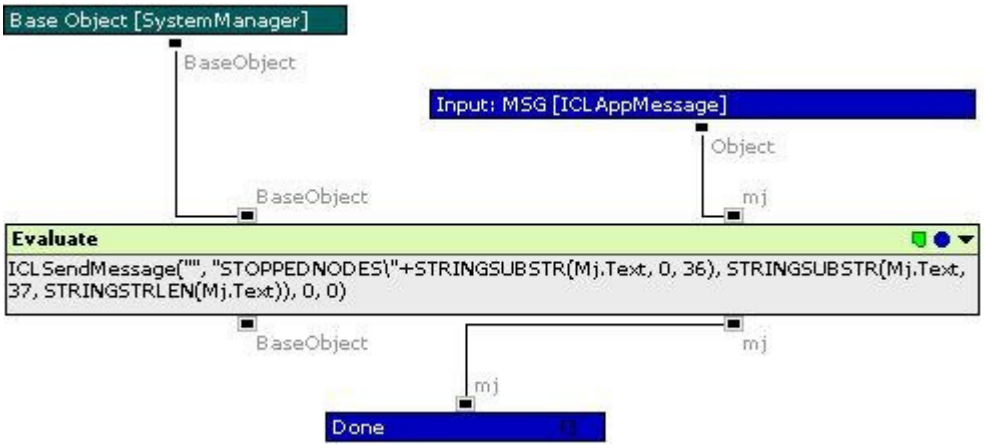

Figure 4.12. WF Message Dispatch Algorithm

In the previous algorithm, we mentioned about the remote workflow engines sending the results of their operations. When the reply message is received at this specified port, WFMessageDispatch algorithm is run. WFMessageDispatch algorithm gets the messages that come from the WF channel as input and parses the message. The message is divided into two parts; the first 36 character of the message string is assigned as the reference of the message sending server. It is saved at the INSTANCE table relating with the stopped

instance nodes. This part refers to the information about the message sending server such as the IP and the unique code of the server.

The remaining part of the message string is assigned as the data related with the workflow item and it is executed via the Evaluate node. In the Evaluate node, Send Message function activates and the message is sent to the channel, named as STOPPED NODES which was created for the workflows in the ICRON system.

When the reactivation message that is carrying the result of the remote process execution is received at the STOPPEDNODES channel, ICRON system executes the ProcessMessage function that is defined as follows:

```
void ProcessMessage (const ICLString& DestinationChannel,
                     const ICLString& ProcessingChannel,
                     ICLOBJECT* pSender,
                     const ICLString& str,
                     ICLOBJECT* pObj)
{
if (DestinationChannel == STOPTicketID)
{
    ReturnMessage = str;
    pNode->CancelSTOPInInstance(iRunInstance);
    if (!pNode->GetAlgorithm()->IsDebugInstance(iRunInstance))
    pNode->GetAlgorithm()->GetRunInstance(iRunInstance)->RunAlg();
}
};
```

Figure 4.13. Process Message Function

Process Message function which is represented in Figure 4.13, calls the parameters required for processing such as Destination Channel, Processing Channel, Sender of the message and the Message itself. The function compares the Destination Channel with the StopTicketID of the stopped node weather it is coming to the right server for processing or not. If the message is received at the right server, CancelStopInInstance function is

executed for the stopped instance in order to reactivate the instance. CancelStopInInstance function, changes the value of the IsStopped attribute from true to false meaning that the node is not stopped anymore and ready to run. Using the NodeUniqueId in the Execution Queue, Run Instance Data that the node was carrying when it stopped is obtained from the database tables and set to the related link points. When all the input link points are loaded with their required data, the node changes its status from “waiting” to “ready to run”.

If the node is a debug instance, meaning that the algorithm is run with debugging and all nodes are run one by one with human intervention, we don’t need to do anything to stop or reactivate the node. As all the nodes are executed according to debugging order, when the turn comes to the “WFSendTCPMessage” node, these expressions are executed one by one and the process is run.

If the node is not run with debugging, the algorithm that is going to run the instance is queried from the database and this algorithm is given to the RunAlg function as a parameter. Knowing the unique id of the node that is stopped, one by one all the tables are read and the model, algorithm, instance and StopTicketID of the node are found. The input link points of that node are also retrieved from the InputLinkPoint table. As the data that the link point carries at the stopping moment is written in the table, it can be obtained for starting the process from the point it stopped. RunAlg function opens the given algorithm, finds the related node, loads the input link point data and by this way reactivates the instance from the point it stopped. The technical implementations of these operations are detailed in Chapter 4.2.8.

Once the instance is reactivated, “ALGINSTStopRun” special function finishes the process of RunAlg function. This function deletes the instance from the execution queue and by this way it unregisters the instance from the listening channel.

4.2.7. Saving the States of the Workflow Algorithms

When the running process is stopped, the state of the running algorithm is saved in the database for reactivation. Workflow algorithms are written within ICRON in order to save the states of the models, algorithms, instances, nodes and input link point data at the stopping moment. The first algorithm defined as WriteAllWFAlgorithmStates which is used to save the states of the workflow algorithms that are run in the model is represented in the Figure below:

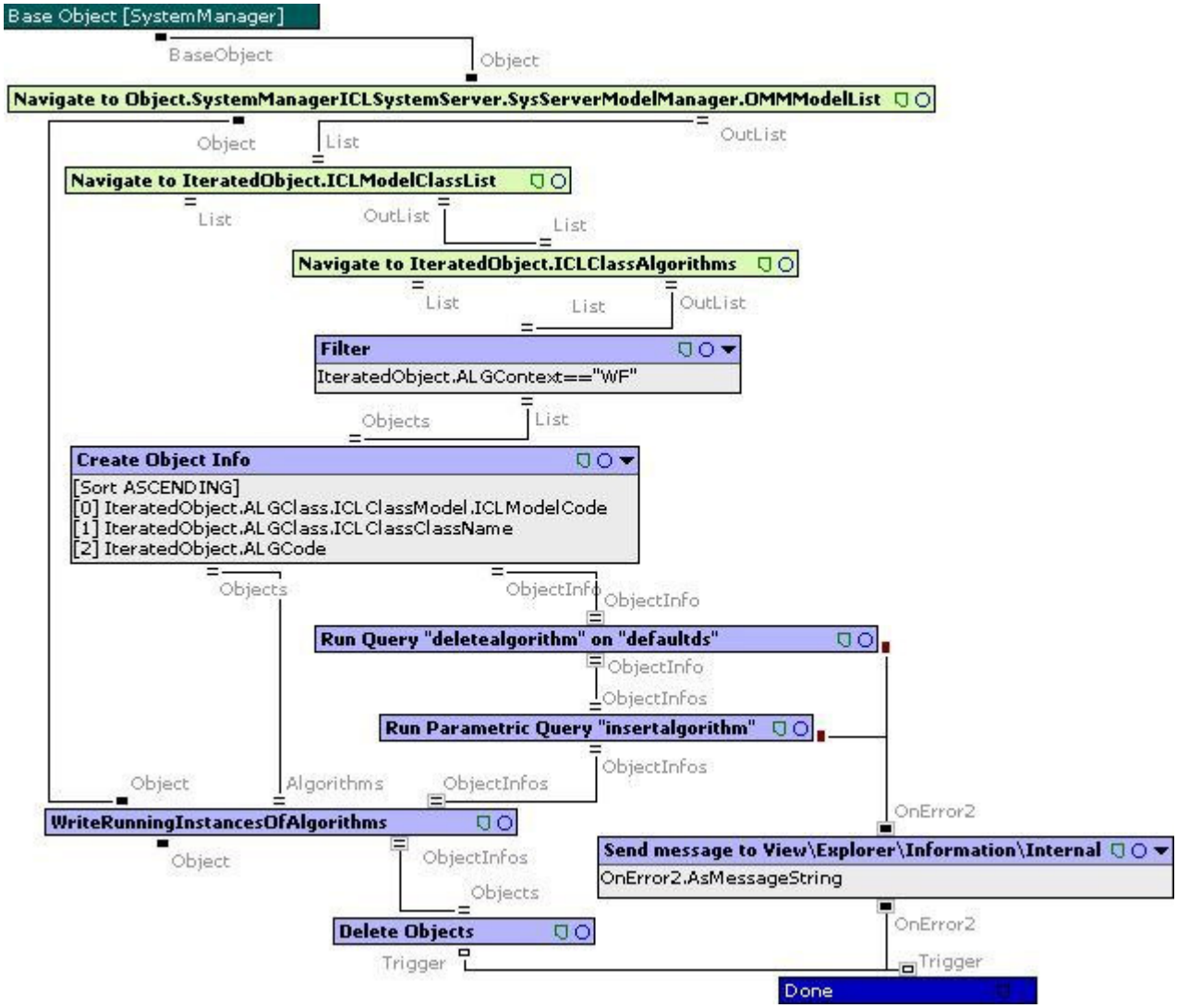

Figure 4.14. Saving the Workflow Algorithms in the DB

In the "WriteAllWFAlgorithmStates" ICRON algorithm, the first Navigate node returns the list of Model Classes.

The second Navigate node returns the list of Algorithms in each Model Class. ListFilter node, checks the list of algorithms and filters the ones that are related to the workflow management system. This attribute is checked by the context of the algorithm. If the context of the Algorithm is stated as WF, then the algorithm is said to be about workflow management system.

When the filtered list of the algorithms is obtained, it is given to the Create Object Info node from an input link point. In this node, the list of algorithms is analyzed and a sorted list of Algorithm Codes is acquired.

The object info that comes from the output link point is given to the Run Query node. This node deletes the data that are written in the Algorithms table in order to prevent duplicate entries. If an error occurs during the delete operation, an error message is given in the Information/Internal Messages Explorer.

When the delete operation is completed successfully, the object info is transferred to the Run Parametric Query node. This node takes the SQL name which executes the insert query that writes the object info into the Algorithms table. If an error occurs during the delete operation, an error message is given in the Information/Internal Messages Explorer. Delete Objects node activates after the insert operation and clears the memory from temporary objects that are written to the databases.

When the insert and clearing operation is completed successfully, the object info is transferred to the next node which will activate the “Write Instances” algorithm. But this node requires two other input data. The first one of them is the list of algorithms. The “Write Instances” algorithm needs this data in order to find the instances of the algorithm. The second data it is requiring is the base object which will give the base Model. Without the base object, an algorithm can not process. As these data can be obtained from the beginning, “WriteRunningInstancesOfAlgorithms” algorithm is ready to activate.

The second algorithm “WriteRunningInstancesOfAlgorithms” is prepared for finding and saving the instances of the running algorithms. Figure 4.15 is a screenshot from ICRON, showing the “WriteRunningInstancesOfAlgorithms” algorithm.

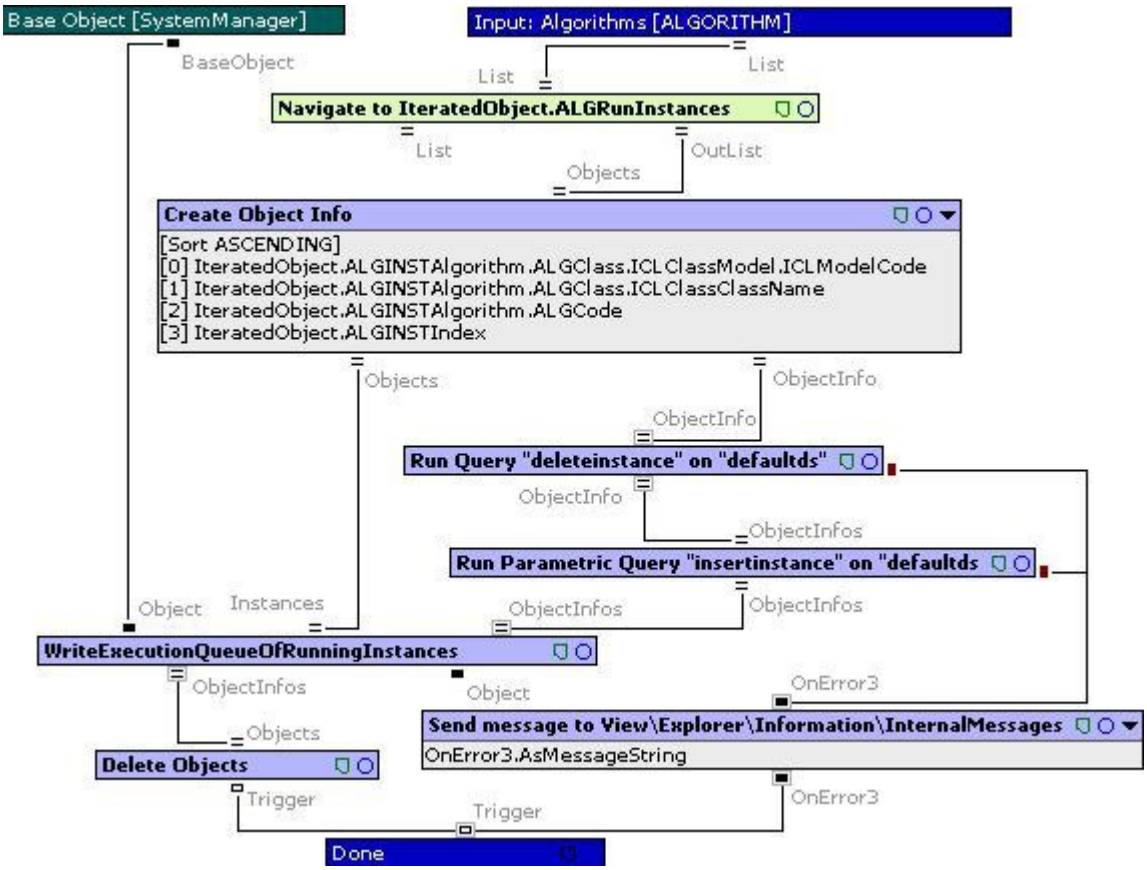

Figure 4.15. Saving the Running Instances of the Algorithms to DB

This algorithm gets the base object as the model itself and the list of algorithms as the Input. The list is given as an output link point data to the next Navigate node. This node returns the list of algorithm instances that are running at the moment. The out coming list is given to the node which will activate the “WriteExecutionQueueOfRunningInstances” algorithm as an input data and Create Object Info node. Similarly, this node analyzes the list of instances and a sorted list of Algorithm Codes and Instance Indexes is acquired.

The object info is given to the Run Query node. This node deletes the data that are written in the Instances table in order to prevent duplicate entries. If an error occurs during the delete operation, an error message is given in the Information/Internal Messages Explorer.

When the delete operation is completed successfully, the object info is transferred to the Run Parametric Query node. This node takes the SQL name which executes the insert query that writes the object info into the Instances table. If an error occurs during the delete operation, an error message is given in the Information/Internal Messages Explorer.

When the insert operation finishes, Delete Objects node activates and clears the memory from temporary objects that are written to the databases.

After the insert and clearing operation, the object info is transferred to the next node which will activate the “WriteExecutionQueueOfRunningInstances” algorithm. When the three input data; List of Instances, Object Data of Instances and the Base Object; the algorithm needs for activating are gathered, the algorithm changes its state from “Waiting to be started” to “Running” and it starts its process. Figure 4.16 represents “WriteExecutionQueueOfRunningInstances” algorithm.

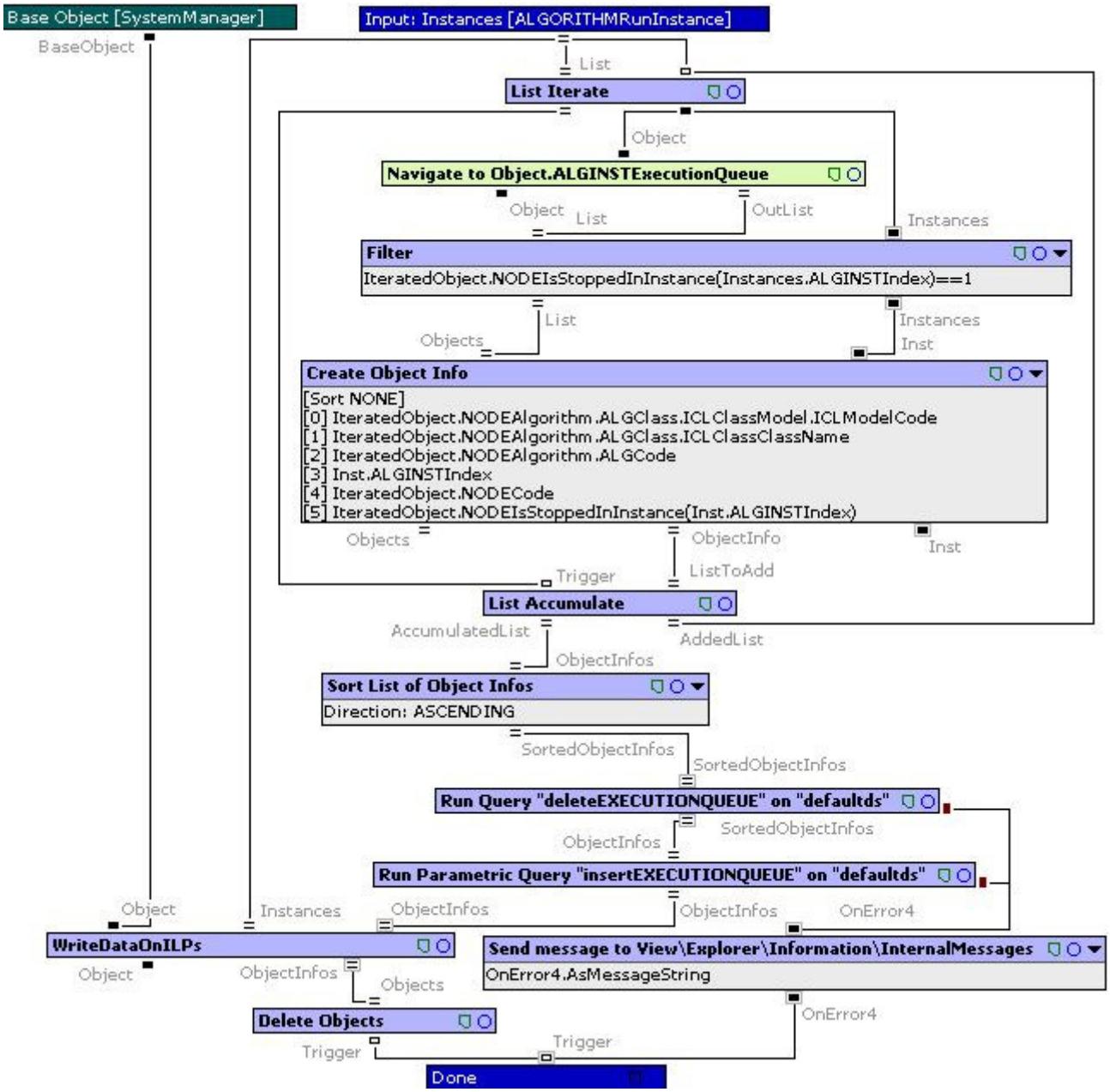

Figure 4.16. Saving the Execution Queues of the Running Instances in DB

The "WriteExecutionQueueOfRunningInstances" algorithm takes the list of algorithm instances as Input.

List Iterate node defines an iteration for all objects in the algorithm instance list. Each instance is added to the process one by one. Navigate node returns the list of instance nodes that are in the execution queue. They are filtered and checked whether they are stopped or not.

When the outlist of stopped execution queue nodes is given to the Create Object Info node, all the required attributes of the Nodes table are taken as output. The model code, class name, algorithm code, instance index, node code and “Is Stopped” attribute of a stopped node is retrieved by this way. One by one all the execution queue is processed and the outlists are gathered at the List Accumulate node.

The list of stopped nodes at the execution queue is sorted at the Sort List of Object Infos node.

When the object infos are ready, they are transferred to the Data Source Manipulation nodes for deleting the table and inserting into the Execution Queue table. During these operations, if an error occurs, messages are given at the Internal Messages Explorer.

When the insert operation finishes, Delete Objects node activates and clears the memory from temporary objects that are written to the databases.

After the insert and clearing operation, the object info is transferred to the next node which will activate the “WriteDataOnILPs” algorithm. When the three input data; List of Instances, Object Data of Nodes and the Base Object; the algorithm needs for activating are gathered, the algorithm changes its state from “Waiting to be started” to “Running” and it starts its process. Figure 4.17 illustrates “WriteDataOnILPs” algorithm.

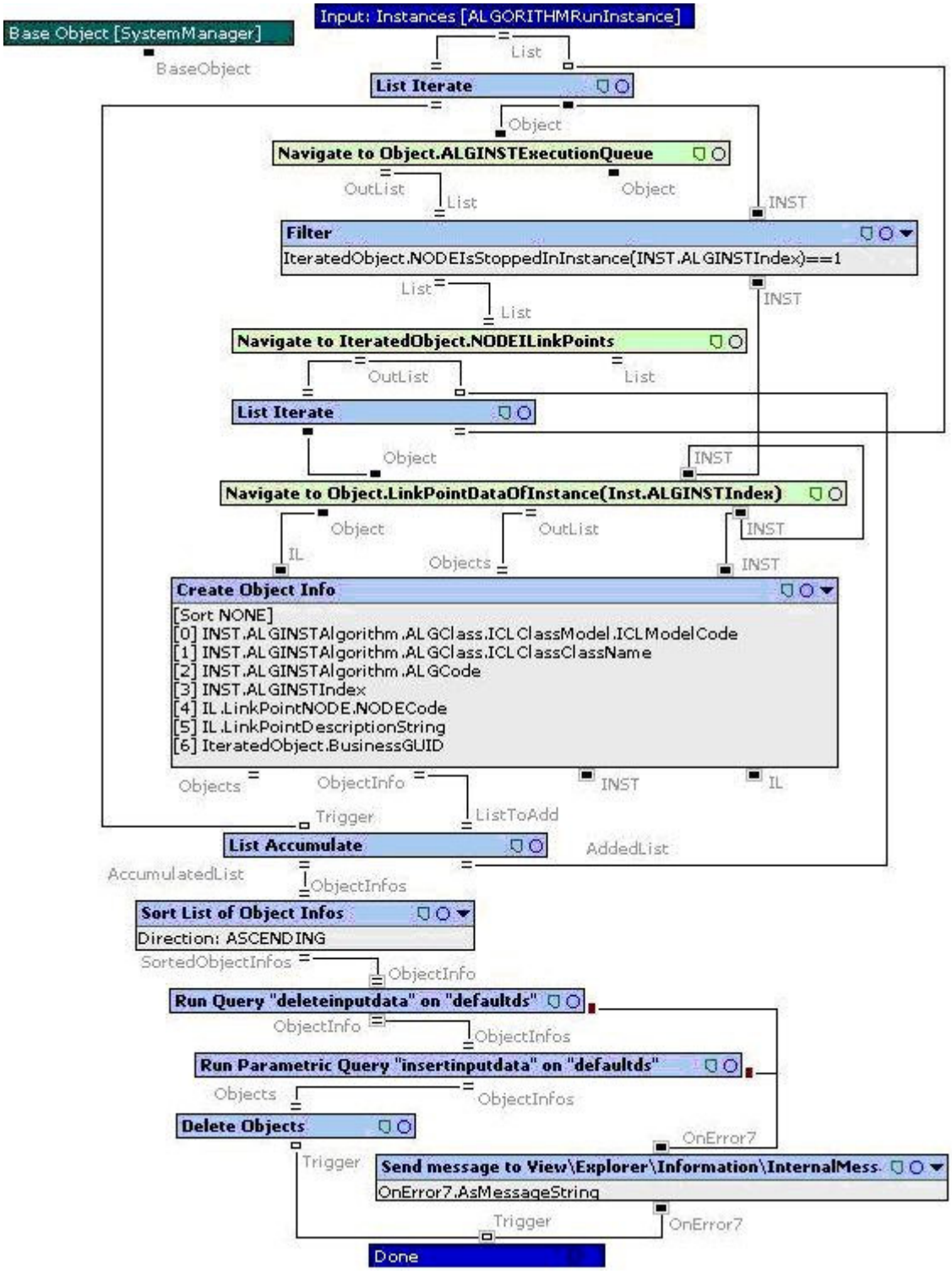

Figure 4.17. Saving the Input Link Point Data of the Stopped Nodes in DB

Similarly, the “WriteDataOnILPs” algorithm takes the list of algorithm instances as Input.

List Iterate node defines an iteration for all objects in the algorithm instance list. Each instance is added to the process one by one. Its model code, class name, algorithm code and instance index is retrieved by this way.

Navigate node returns the list of instance nodes that are in the execution queue.

List Filter node checks the Stooped attribute of the instances in the execution queue and filters the ones that are stopped.

The second Navigate node returns the list of input link points that belong to the stopped instance node in the processing execution queue.

The outlist is iterated by a second List Iterate node in order to get the input data that are carried on the input link points of the stopped instance node. The third navigate node returns the input data of the given input link point that is carried at the stopping moment. With the iteration process, the list of input data is gathered and transferred to the Create Object Info node.

All the other required input data is also given to the Create Object Info node. The list of the Input Link Points are taken from the object that is reached by the second navigate node. Lastly, the list of Instances is taken from the previous algorithm as Input.

When all the three input data are gathered, the Create Object Info node activates and returns the input link point's description string, node code and BusinessGUID as the reference data of the stopped instance's input link point.

One by one all the execution queue is processed and all the input link point data that belong to the stopped instance's nodes are gathered at the List Accumulate node.

The list of input link point data is sorted at the Sort List of Object Infos node and they are transferred to the Data Source Manipulation nodes for deleting the table and inserting into the Input Data table. During these operations, if an error occurs, messages are given at the Internal Messages Explorer.

When the insert operation finishes, Delete Objects node activates and clears the memory from temporary objects that are written to the databases.

When all the nodes finish their processes, the Done node of this algorithm turns into green showing the successful finishing of the algorithm run. “WriteDataOnILPs” algorithm’s state changes to “Finished” and the Done node of the “WriteExecutionQueueOfRunningInstances” algorithm becomes the next step to be processed. When “WriteExecutionQueueOfRunningInstances” algorithm also finishes, the Done node of the “WriteRunningInstancesOfAlgorithms” algorithm activates. Lastly, when it is finished, the turn comes to the Done node of “WriteAllWFAlgorithmStates” algorithm. Here, when the last Done node is processed, the total saving algorithm finishes. When the system receives a remote process message, stating that the algorithm is ready for reactivation, the state of the instances and the data they are carrying at that moment are read from the database tables and they are put into the execution queue again for processing from the point it stopped.

4.2.8. Creating a New Instance and Loading it to the Execution Queue for Reactivation

The main problem in workflow management systems are the loss of states and data during breakdown or stopped instance cases. In order to fix this problem, we prefer to save the instances that are run in the execution queue. Hence in a stopping case the system can read the last node processed in the execution queue from the database tables and reactivate it after setting its data.

The following “LoadRunInstancesOfWFsFromDB” algorithm in Figure 4.18 finds the last processed instance saved at the execution queue table and creates a new instance in order to set its data and reactivate.

The first “Run Select” node executes the “readinstance” SQL statement and returns all the rows in the Instance table. This table consists of the Model codes, Class Codes, Algorithm codes and the Instance Indexes of the stopped processes.

On the other hand, the algorithm codes that exist in the system manager model are obtained by the navigate nodes. The first “Navigate” node returns the list of models contained in the System Manager. The second “Navigate” node returns the list of the classes in the model classes. The third “Navigate” node returns the list of algorithms that are connected with the chosen model’s classes. “List Filter” node is used for filtering the Workflow algorithms.

The instances that are read from the database are related to the algorithms which are found by the navigation process. If the instance in the database belongs to the same algorithm that is found by the navigation process, it is set as the children of the parent algorithm.

All the instances are iterated and their algorithms are found by the Navigate to Object node. When the parent object is navigated by casting the algorithm, it is checked whether it still exists or not. If the algorithm exists, it is transferred to the Assign node for creating a new instance. A new instance is created with the StartRunInstance function for the transferred algorithm and its instance index is saved at the New Instance Index local field.

After the assign process, next Navigate node finds the list of Instance Indexes belonging to the transferred algorithm. The instance indexes are compared with the new instance index saved in the local field at the “Pick Object” node. If the new instance’s index is the same as the model’s algorithm instance index, it is transferred to the Assign node through the INS input link point. The fourth column in the Instance table holds the Instance Indexes. As the GetNumber function starts counting from 0, the fourth column of the Instance table is found by the OI.GetNumber(3) statement. By this way, the saved instance indexes are obtained. And the new instance index is related to the old instance index saved at the instance table. The old instance index is assigned as the previous index

of the new instance and it is transferred to the “List Accumulate” node for gathering the Instance Indexes’ list.

When all the new instances are created and related to the saved instances in the database, they are transferred to the LoadDatatoActiveILPsFromDB algorithm for picking the input link point data of the stopped instances that are used for reactivation.

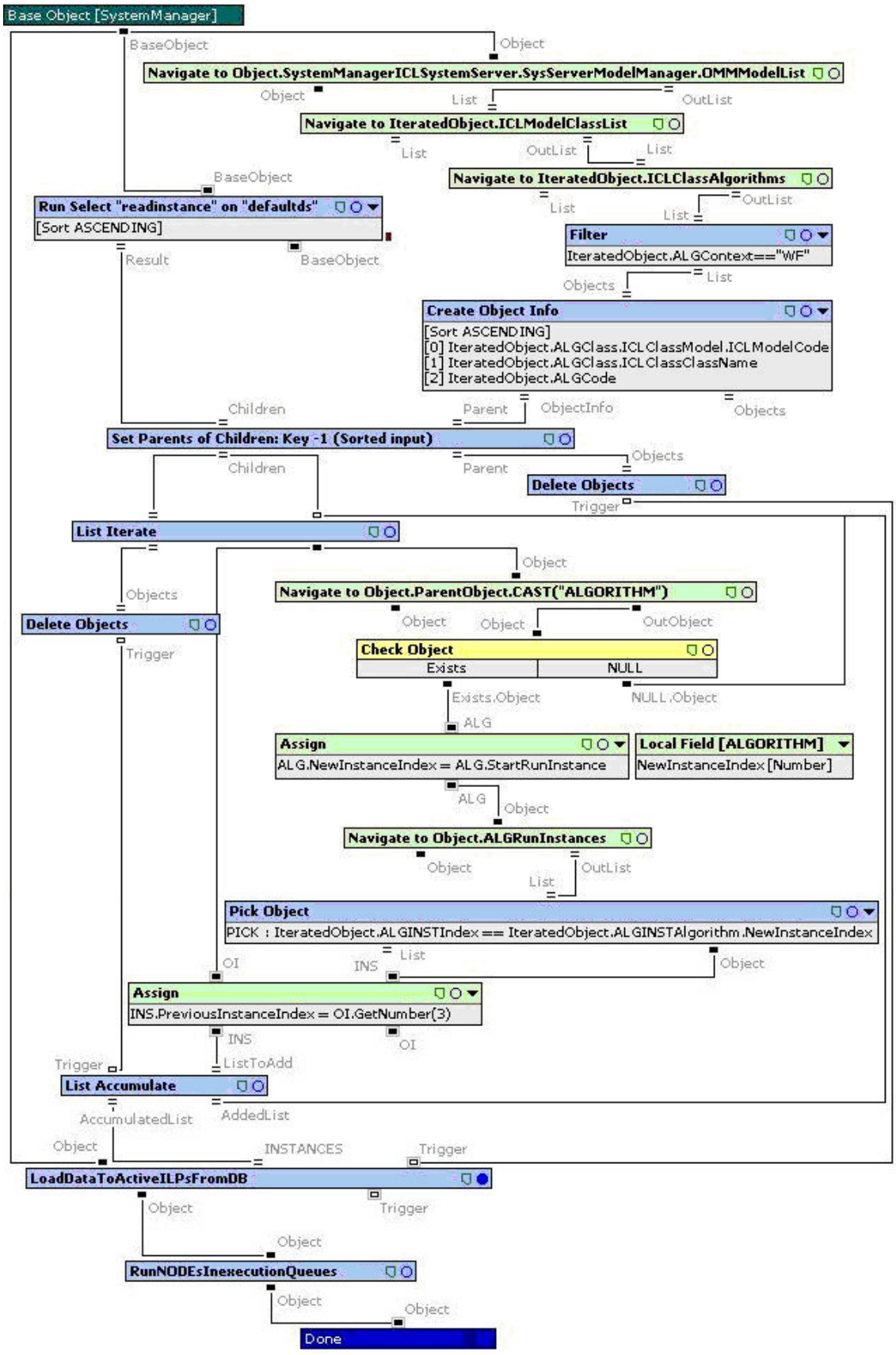

Figure 4.18. Creating a New Instance

4.2.9. Setting the Input Link Point Data for Reactivation

LoadDatatoActiveILPsFromDB algorithm takes the list of instances to be reactivated as input and compares the instance indexes with the ones in the Input Data table.

"Run Select" node reads the rows in the Input Data table. Create Object Info node returns the model codes, class names, algorithm codes and the previous indexes of the instances that are to be reactivated. The instances of the input link point data that are read from the input data table are related to the instances that are found by the create object info node. If the input link point data in the database belongs to the same instance that is found by the create object info node, it is set as the children of the parent instance.

All the input data are iterated and their instances are found by the Navigate to Object node. When the parent object is navigated by casting the algorithm run instance, it is checked whether it still exists or not. If the instance exists, first of all its algorithm is found and then its node list is found via navigate node. The second navigate node returns the list of input link points of these nodes. Since there may be several nodes in an algorithm and several input link points of these nodes, a last "Pick Object" process is needed.

The instances that their data will be set are chosen in the "Pick Object" node. In this node, the record in the Input Data table is analyzed, its fifth column, Node Code is get via OI.GetString(4) statement. It is checked whether it is the same as the stopped instance's node code found via iterated objects or not. Also the fourth column in the Input Data table is analyzed and the Link Point Description String that the node carries its data on it is obtained via OI.GetString(5) statement. It is also checked whether it is an input link point of the stopped instance's node. If the appropriate data is picked up by the Pick Object node, it is transferred to the Get Business Object node for finding the reference data.

Get Business Object node finds the seventh column of the Input Data table record is obtained by OI.GetString(6) statement. This string carries the reference data which was held on the input link point at the stopping moment. As the node and the input link point of the stopped instance is found, this reference can be converted to a business object by the

Get Business Object algorithm and it is set as the data of the chosen instance in the Evaluate node via the Set Data function.

One by one each instance is analyzed and its stopped node and input link points that were carrying data are found. After their data are converted to business objects and evaluated, the data setting algorithm is finished.

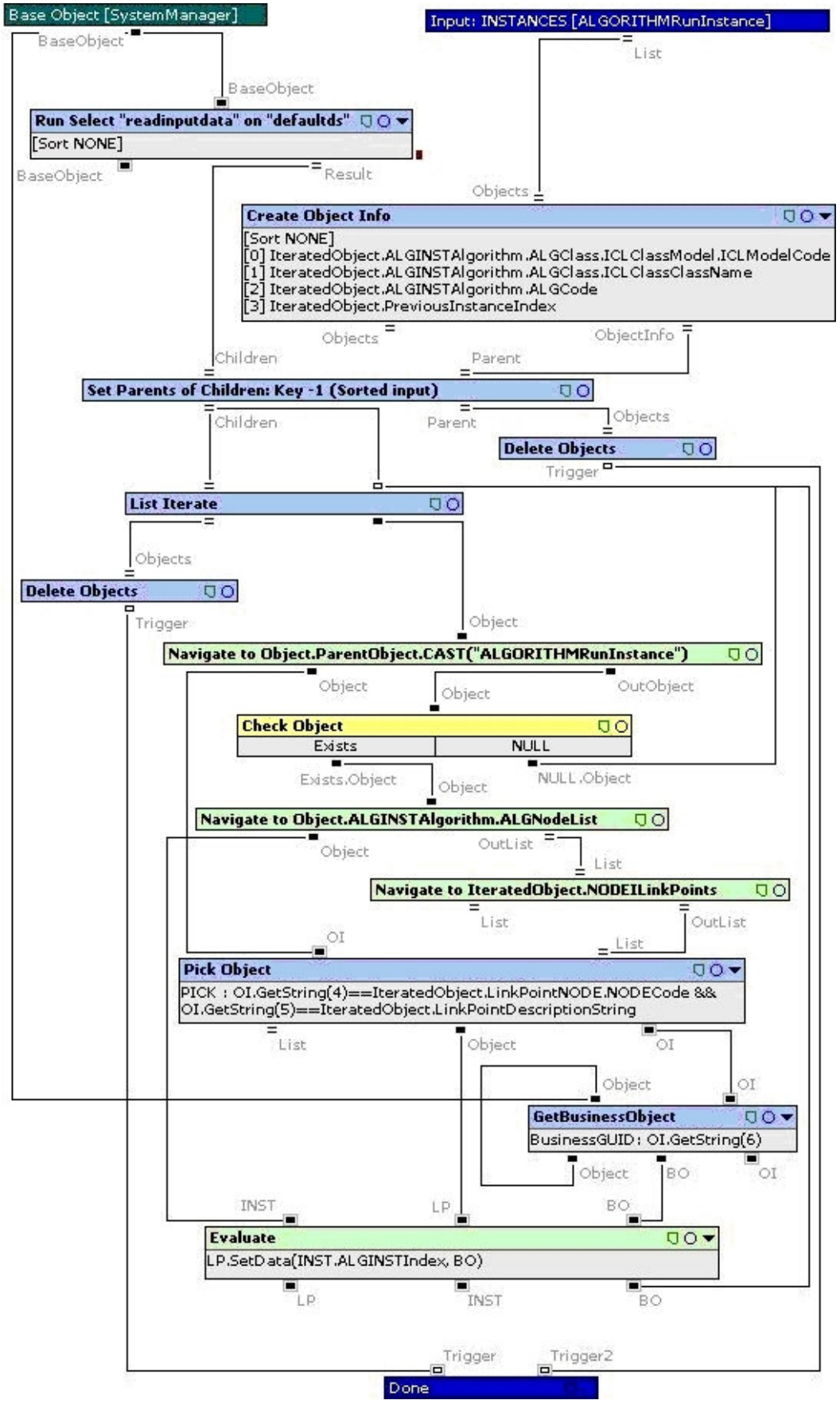

Figure 4.19. Loading the Stopped Instance's Input Link Points

As the LoadDatatoActiveILPsFromDB algorithm finishes, LoadRunInstancesOf WFsFromDB algorithm also comes to the last step. While all the data are set to the input link points of the instance nodes, they change their status to “Ready to activate” mode. Having the instances ready to be reactivated and their data set, the last node in this algorithm runs the following RunNODEsInExecutionQueues algorithm.

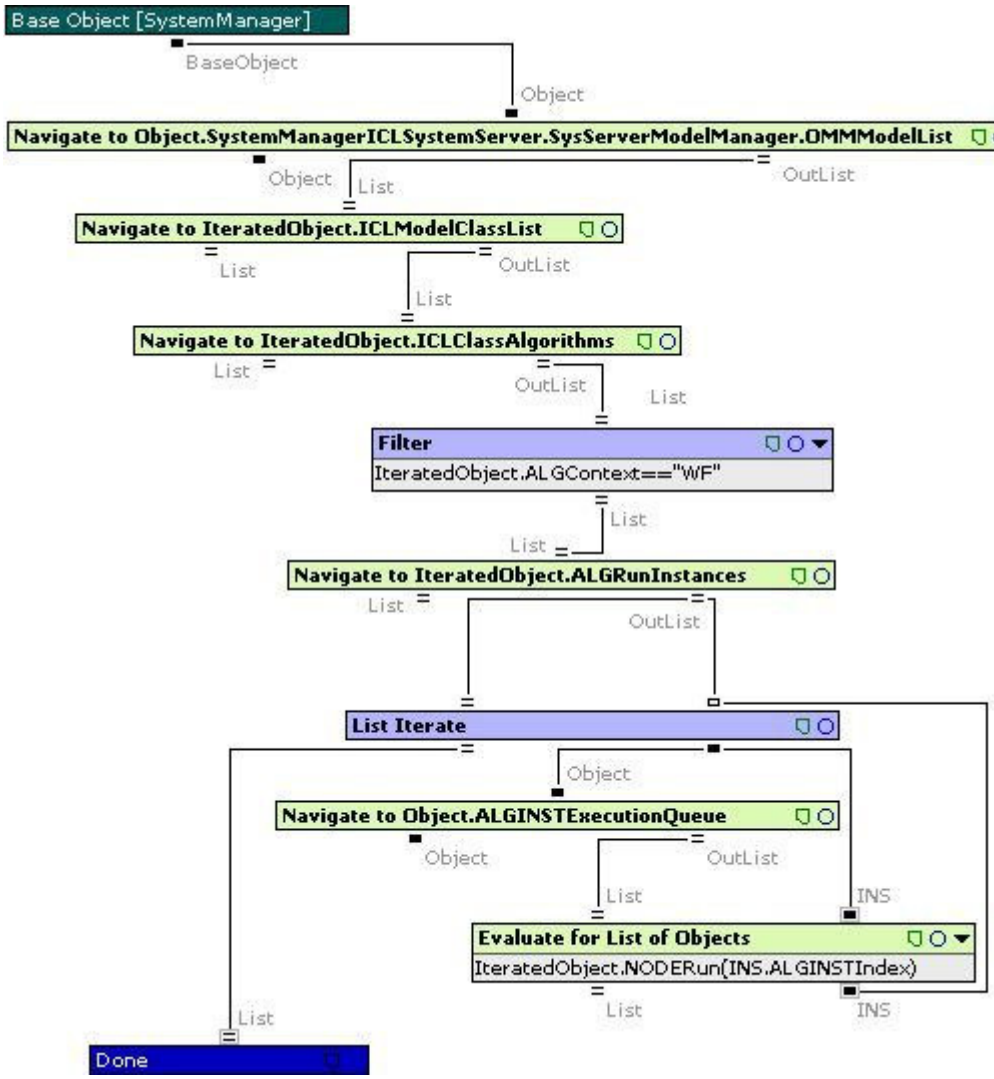

Figure 4.20. Running the New Instances in EQs

This algorithm finds the model list contained in the system manager by the first navigate node. The second navigate node finds the list of classes in the models and the third navigate node returns the list of algorithms in the classes. All the algorithms are filtered via the Filter node and checked whether they are related to the workflow system or not. The next navigate node returns the list of run instances belonging to the navigated list

of algorithms. Lastly, the list of execution queues is iterated and they are transferred to the Evaluate node. In the evaluate node, Node Run function runs the instances that are in the execution queue one by one. When this algorithm finishes, the turn comes to the last Done node of the LoadRunInstancesOfWFsFromDB algorithm.

With the last Done node finishing its process, a new instance will be created in order to reactivate the stopped instance and its data will be set.

5. A SIMPLIFIED ORDER RELEASE EXAMPLE

In this work as an example of a distributed workflow management system, an order release procedure is given. Figure 5.1 represents the flow of our order release procedure:

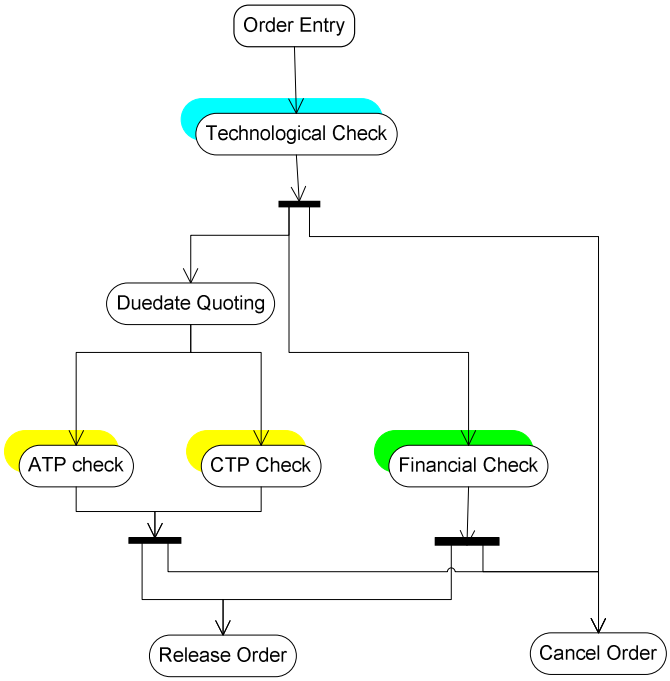

Figure 5.1. Workflow of an Order Release Procedure

When an order of a product is received, firstly it is checked by the technological controllers. If the order is producible, it is sent to the planning department for due date quoting and accounting department for financial check. Each control operation is performed by different departments and they all have internal control workflows. Each operation waits for its predecessor operation to finish its control and starts its work according to the answer coming from the previous task. For example in our workflow, due date quoting operation does not start unless the technological check operation ends with a result approving the production, showing that the order is producible. This usually happens in factories producing customized materials like guy. Sometimes customers order such materials that are so thick, thin or long to be produced according to the standards. In these cases, orders are canceled or changed with producible materials.

When the technological approval is received, planning and accounting departments start their work. Accounting department checks whether the customer is risky or not by controlling its credit status. Planning department gets the quantity and the specs of the product, and starts ATP (Availability to Produce) and CTP (Capability to Produce) controls. These control mechanisms, that are included in the ICRON system, check whether the required amount of the ordered product is available at stock or can be produced till the given due date. Planning department may return an available stock quantity or propose a new due date. Or, they can accept the release of the order with the given quantity and due date using the products in the stocks. In each case the control operations end with the cancellation or the approval of the order.

Each control algorithm may run on separate remote platforms. The communication between these distributed systems is supplied through messaging technologies. ICRON can send TCP messages, SOAP messages; it can directly connect to the remote ICRON systems or send messages through MQs. In our example all the control workflows performing their tasks send TCP messages to the listening remote systems. Since each remote system is used by different departments, they all use a customized module of ICRON like planning or accounting, and they all have specific ports opened for listening messages coming from the predecessor departments. At the initialization stage of ICRON, these ports are opened and registered in order to listen the incoming messages.

Since all the remote servers have to know each other, they are registered as Actors in each operating system's database. These databases save the names and the ports of each operating system (ICRON systems in our example) at tables named as ACTOR. These tables contain the code of the working ICRON module, the IP of the running ICRON server and the number of the port that is registered for getting the messages. These attributes are carried on the ACTOR object of each ICRON system manager. When an ICRON system that runs the first task of the workflow is initialized, the ACTOR objects are created. Having the ability to reach each ACTOR through its name, remote servers do not need to know the IP and the port of the ICRON system that it is going to send approval or cancellation message. It just finds the IP and the port of these following operating systems by looking at the ACTOR tables for finding the "Planning ICRON Actor" or "Financial ICRON Actor" and sends the message.

The following algorithm shows the main workflow of the order entry operation. Each sub algorithm runs separate workflows.

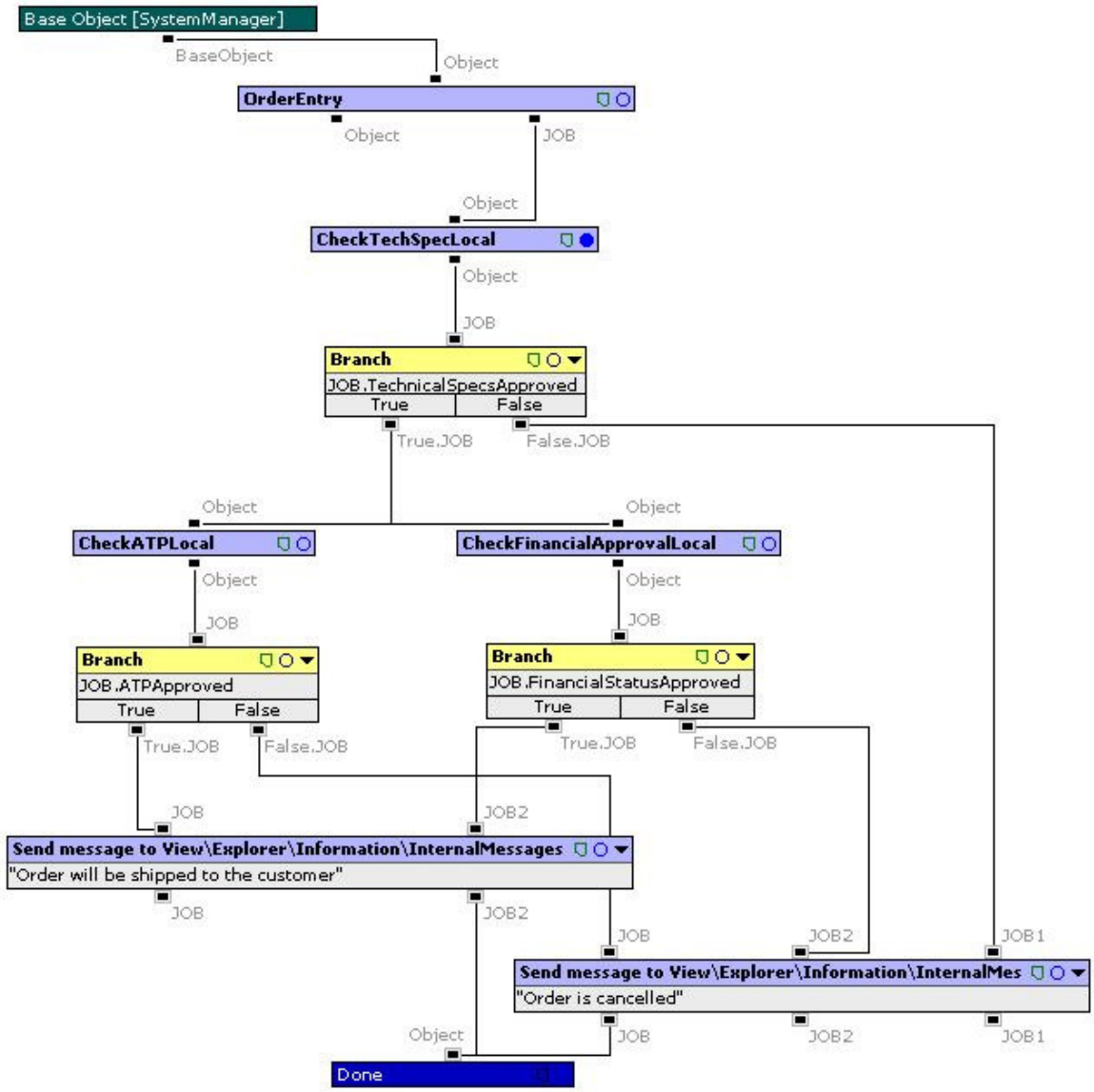

Figure 5.2. Main Workflow

When a customer starts to enter his/her order from an online system, the “Main Workflow” algorithm starts to execute in the ICRON system. The first node of this algorithm, OrderEntry node, runs a separate algorithm that opens a popup for entering the details of the order. In this example, the code of the ordered part, the size of the batch needed at the given release time, until the given duedate is saved as a Job Object as represented in the following figure illustrating the Order Entry algorithm.

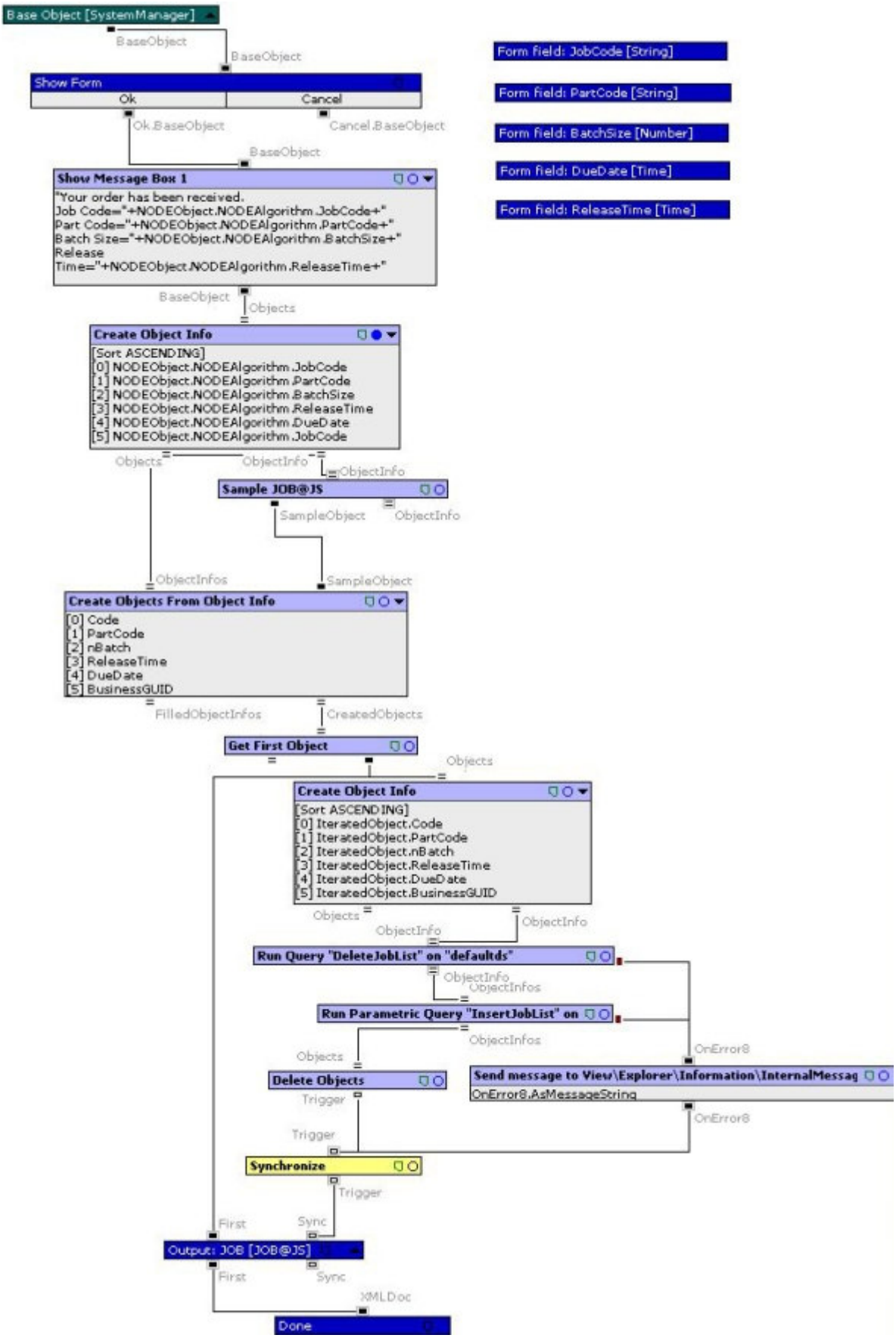

Figure 5.3. Order Entry Algorithm

As the specifications of the order is taken and saved as a Job Object in the ICRON system, it is transferred to the related departments for a series of controls. The first of these control mechanisms is executed with the second node in the Main Workflow algorithm. CheckTechSpecLocal node fires the local workflow that checks the producibility of the received order. The following algorithm shows the flow of the local controlling procedure.

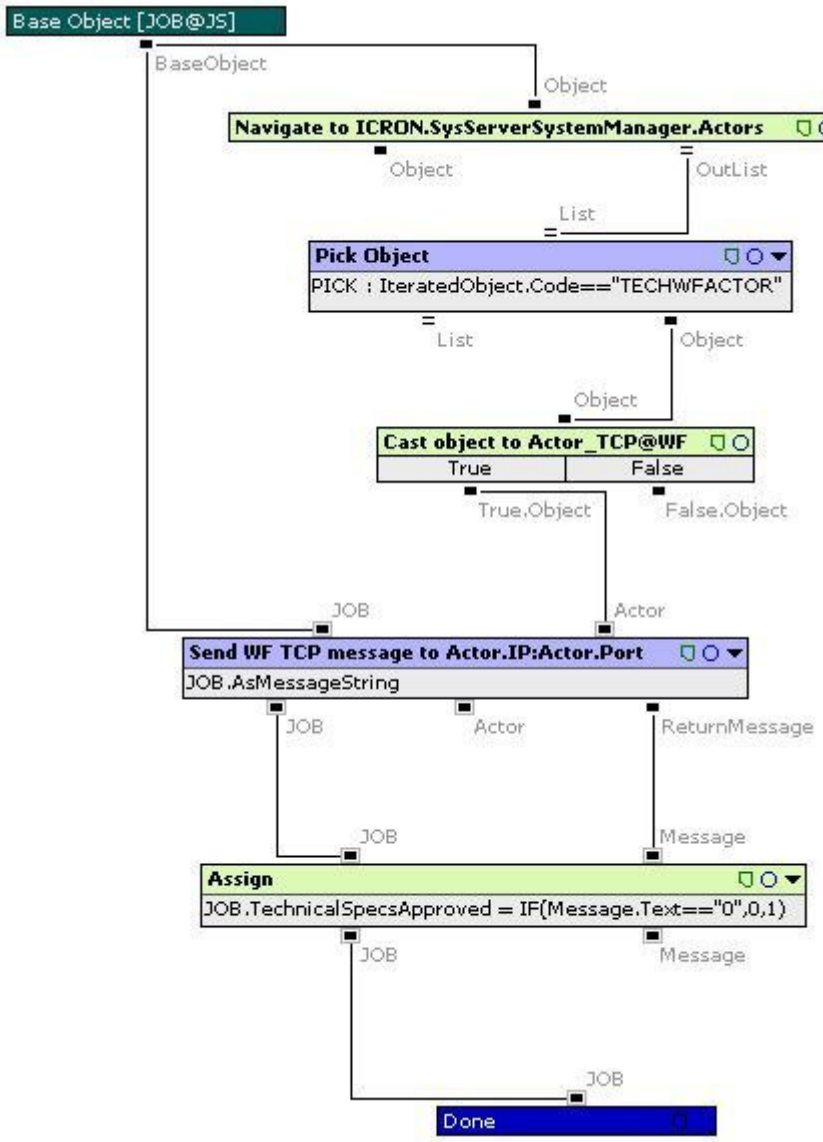

Figure 5.4. Technical Specification Control Workflow Execution

As the Main Workflow is run on the Main ICRON System, the controlling workflows are run on distributed servers that are connected with each other. In this example, the created Job Object carrying the specifications of the order is transferred to the

local server which has a registered Actor named as TECHWFACTOR. The main ICRON system reads the ACTOR table and finds the IP and the Port of the TECHWFACTOR server that is waiting for the messages transferring the Job to be checked technically. When the ACTOR is found, the Job is sent to the local server for control with the newly created "Send WF TCP Message to Actor.IP:Actor.Port" node. This node is created within the context of this thesis for sending the tasks of WFs to other distributed systems. The description and functionality of this node is detailed in Chapter 4.2.4, Stopping an Instance. Since the Job is sent to the local server for technical control, the Main Workflow has to stop and wait for the result of technical producibility check. Therefore "Send WF TCP Message" node firstly stops the running "Main Workflow" algorithm and creates a STOPTICKETID carrying the details of the server that is running this algorithm. Also the state of the stopped algorithm and the STOPTICKETID of the main server are saved in the database for retrieving when the result is received and the algorithm is to be run from the point it stopped. After saving the state to the database, the main server that is waiting for the result of the technological check, registers itself to the STOPPEDNODES channel of the main ICRON system using its STOPTICKETID, sends the Job Object as a message to the technical controlling ICRON system and starts to wait for the returning message from the local WF Actor. Through this time, the "Send WF TCP Message" node turns into yellow in color representing that it is waiting for an input object like the incoming message for reactivation.

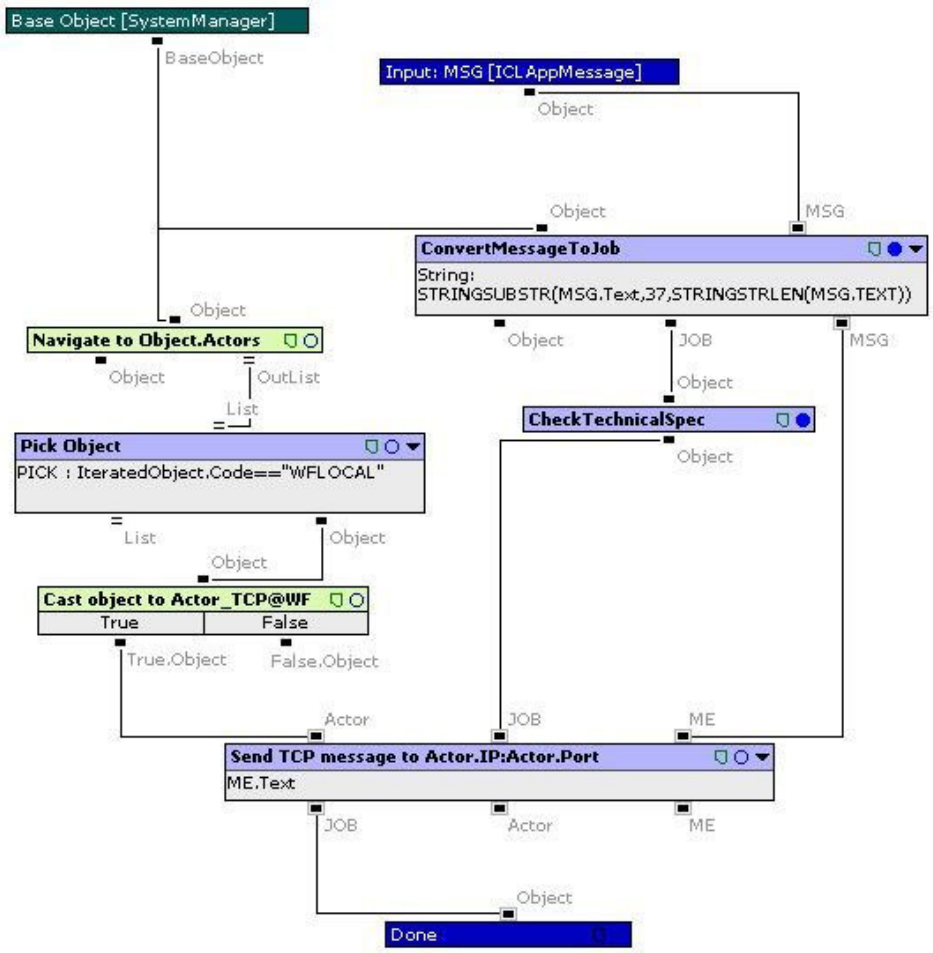

Figure 5.5. Remote Technical Spec Controlling System

When the job is received at the TechWFActor's listening port, "Check Technical Spec WF" algorithm which is represented above is executed at this remote server. The Input message string is triggered and starting from the 37th character, the message string is converted to a Job Object. The first 36 character of the message string carries the STOPTICKETID of the message sending server. Having the Job Object, Check Technical Spec procedure starts, the details of this procedure is not important in the context of this thesis, only the result of the control operation is important. What is controlled during the technical spec check is not known, only the result of 1 or 0 that is returned to the Main Workflow depending on the specifications of the ordered part is known.

When the Check Technical Spec node finishes its task, it sets the value of the "TechnicalSpecsApproved" attribute of the job to 1 or 0 on the remote server and sends the job as a message to the Main Workflow through Send TCP Message node.

Since the Main Workflow gets the return message that it is waiting for, the “Send WF TCP Message” node reactivates and turns into pink in color as explained in Section 4.2.6 representing that it is ready for execution again. As this node’s task is finished, the Assign node is activated as shown in Figure 5.4. The returning message is triggered and the last part carrying the result of the technical specification control is assigned at the “TechnicalSpecsApproved” attribute of the job as 1 or 0 on the main server. When the Check Tech Spec Local algorithm is ended, the branch node in the Main Workflow starts its task. It checks the value of the assigned “TechnicalSpecsApproved” attribute of the related job. If the value returns 1, operation goes one step ahead and the Job is sent to the Financial and Planning departments for approval. If the value returns 0, the control operation is ended and a message showing that the order will be cancelled is sent to the customer and related departments.

Similar controlling algorithms are run on remote servers in order to check the availability or capability to produce the ordered material till given due-dates and the trustworthiness of the customer. Similarly, the Main Workflow is stopped when each remote process starts to run and reactivates when the result is received back via TCP messages. The same branching is done after each control procedure is finished and the approval or the cancellation of the order entry process is decided by this branching operation.

6. SUMMARY AND FUTURE RESEARCH DIRECTIONS

In this work, distributed workflow management systems are introduced and a workflow engine which can be used in the competitive and dynamic business environment for managing distributed workflows is proposed. The need for using workflows in business process modeling operations is explained in the work by referencing the published literature and in contrast, the insufficiencies of the existing workflow systems are detailed.

Workflows are used in business operations since they are better managed and monitored than the manual operations, but they are usually managed using a single central server which increases the number of failures. Using a single server in management, the workflow engines can communicate successfully but the communication traffic causes the server to be overloaded. Therefore using several servers for managing the workflows is preferred. According to two approaches, mirror databases can be used in multi servers or dynamic server transfer system can be developed in order to minimize the impacts of failures in single server systems. But there are also some other handicaps of using multi servers in workflow management. Since several identical database servers are used in mirroring for the management of a workflow, the definition of the workflow has to be included in each server. When a change is made in the definition of the workflow, all the definitions in the database servers have to be changed, causing the servers to be kept synchronous, and the management cost to be increased. In addition, as the workflows running on different servers might be executed in different systems, they might require decoders and encoders to be used for communication. The dynamic server transfer approach allows tasks to be transferred between servers but if the servers use different workflow execution systems, then the transferred task can not be executed without decoding the definition of the task. This also increases the costs and requires more time for execution which is not acceptable for most of the business entities. In order to lower the costs of maintaining multi servers, increase the performance and scalability of the servers, allow dynamic definition changes, we propose the use of a distributed WF engine.

With the implementation of the proposed workflow engine in ICRON system, some important contributions of our work have revealed:

- *Increased Server Performance, Scalability:* Since all the operations are not run by a single server and also the communication operations are not handled on the same server; there occurs no bottleneck cases, the servers and communication channels do not become overloaded. With the distribution of workflow tasks between several workflow engines, the workload on the servers are increased, the servers become scalable and robust.

- *Allowing Dynamic Changes, Flexibility:* Although the management of all worklists is not easy to maintain in distributed systems; since each processing engine holds the definition of its own workflow item, in case of a change in the definition of a workflow item, update of the related worklist is enough. The workflow item definitions need not to be managed by a central server; they can be stored in distributed workflow agents and changed dynamically. This shows the flexibility of the system.

- *Business Data Persistency:* Another additional benefit of the proposed system is the business data persistency. Existing WfMSs lose the business objects in system failure cases. In order to prevent these, in this work, we save each job object in the business data stores of the workflow agents. By this way we can recreate the job objects that are parts of the workflow item, as if they are created at the initialization stage of the system and none of the jobs related to the workflow item are left without processing.

- *State Persistency:* Another handicap of the existing WfMSs is the loss of running instance states. When a task needs an input data for running a process, it stops and waits for the data to be received. This data can be a result of another task processed in another workflow engine. While waiting, the instance must not be lost since the result has to be inserted into the right task. Therefore in this work, we save each stopping task's instance and data it is carrying before stopping, in instance data stores. When the needed input data is received in the workflow engine, the instance

data store is triggered and the stopped instance and its data are retrieved for reactivation. Also there is another contribution of this thesis which can be detailed in this part. When the instances and data are stored and the task is stopped, a message is sent via TCP to remote workflow engines that process the related part of the job. The whole job is not sent through these messages therefore the load of the server is decreased. In addition the messages supply the synchronization of the workflow agents.

Supplying such contributions as listed above, the proposed workflow engine can be used in several workflow agents. In the context of this thesis, we have used messages sent via TCP for communication. Future applications can use other messaging techniques like SOAP or MQ for supplying the communication of workflow agents.

In addition, future researches may try to visualize the object creation operations since these operations can not be followed in the existing workflow engine unless internal messages are sent to ICRON windows. In this work, the object creation operation is automated by an algorithm running at the initialization stage. While this algorithm is running, it sends messages to the internal messages window in ICRON showing that the business objects are created.

Another future research can include a report which can be implemented in order to list the stopped instances in the system. This report can retrieve the stopped instances from the instance data stores and see the status of the workflow execution. Getting the stopped instances via report, their input data can be retrieved and set for reactivation process automatically by running a report.

REFERENCES

1. Aalst, W.M.P. van der and K.M. van Hee, *Workflow Management: Models, Methods, and Systems,* MIT press, Cambridge, MA, 2002

2. Alonso, G., Institute of Information Systems, Switzerland; B. Reinwald, C. Mohan, IBM Almaden Research Center, USA; *Distributed Data Management in Workflow Environments*, 1997

3. Alonso, G. Institute of Information Systems, Switzerland; D. Agrawal, A. El Abbadi, Depatment of Computer Science UC Santa Barbara; C. Mohan, IBM Almaden Research Center, USA; *Functionality and Limitations of Current Workflow Management Systems,* 1997

4. Bauer, T. *Efficient Realization of Enterprise-wide WfMS,* PhD thesis, University of Ulm, Germany, 2001

5. Bauer, T., DaimlerChrysler Research and Technology, RIC/ED; M. Reichert, University of Ulm, Databases and Information Systems Dept., Germany; *Dynamic Change Of Server Assignments In Distributed Workflow Management Systems,* Proc. 6th Int'l Conf. Enterprise Information Systems (ICEIS'04), Volume 1, Porto, Portugal, April 2004

6. Chang, S., and C. Jaeckel, *Oracle workflow guide. A publication of Oracle Corporation*, USA, 2000

7. Ellis, C.A., K. Keddara, G. Rozenberg, *Dynamic Change Within Workflow Systems In Conference on Organizational Computing Systems,* N. Comstock et al. (eds.), ACM Press, Milpitas, 10-21, 1995

8. Georgakopoulos, D., M. Hornick, A. Sheth, *An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure*, Distributed and Parallel Databases, 3, 119-153, 1995

9. ICRON Technologies, http://www.icrontech.com, 2006

10. Kappel, G., S. Rausch-Schott, W. Retschitzegger, *A Framework for Workflow Management Systems Based on Objects, Rules and Roles,* ACM Computing Surveys, 1998

11. Leymann, F., D. Roller, *Production Workflow - Concepts and Techniques,* Prentice Hall, 2000

12. Li, S., Ryerson University; D. Coleman, University of New Brunswick; *Modeling Distributed GIS Data Production Workflow,* Canada, December 2003

13. Meng, J., S.Y.W. Su, H. Lam, A. Helal; Database Systems R&D Center, University of Florida; *Achieving Dynamic Inter-organizational Workflow Management by Integrating Business Processes, Events, and Rules,* Gainesville, Florida, USA, 2002

14. Muth, P., D. Wodtke, J. Weißenfels, A. Kotz-Dittrich, G. Weikum, *From Centralized Work-flow Specification to Distributed Workflow Execution*, JIIS, 10(2), 1998

15. OMG, Object Management Group,1997 http://www.omg.org

16. Schulz, K. and Z. Milosevic, *Architecting Cross-Organizational B2B Interactions,* University of Queensland, Australia, 2004

17. WfMC, Workflow Management Coalition, *the Workflow Reference Model*, http://www.wfmc.org, Document Number TC00-1003, England, January 1995