MilramX Data Sheet

Introduction

MilramX is a Real-Time Artificial Intelligence (AI) platform for integrating the industrial enterprise by "Making sure that everyone in the organization has the information they need to do their jobs, when they need it".

Industrial organizations waste large sums of money due to sales leads not followed up, customers being extended financial terms when they should not be, materials not available when needed for production runs, production operations taking too long, and customer orders incurring late shipment penalties. These and many more problems all result from people in the organization not getting the right information, at the right time, to prevent these problems occurring.

MilramX solves this problem by automatically monitoring data in the various systems used by individual departments within an organization. MilramX then uses AI to turn this into actionable information, in the form of text messages, Emails, or updates to other systems, that people in different "silos" within the organization need and/or that need to be communicated to customers and suppliers.

Management Paradigm Shift

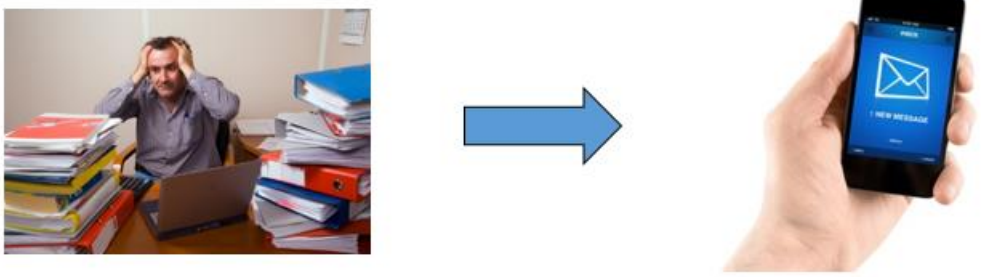
Industrial organizations collect and generate large amounts of data every day in systems used by organizations such as sales, marketing, finance, production, and materials management. In addition, these departments exchange data with suppliers, customers, and logistics organizations.

The problem is that all of this data is typically only available as reports to the people within organizational "stovepipes" who use the systems that generate the reports. Exchange of data between people in different stovepipes is typically done by means of coordination meetings, phone calls, Emails, and paper forms, all of which are extremely inefficient.

 © Copyright Milramco LLC

Even worse, reports from systems used by people in these stove-pipe organizations typically detail what went wrong yesterday, rather than giving people the real-time information they need to head-off problems before they occur today. Even with dashboards and on-screen displays, all a stovepipe system is doing is presenting data in a more easily digestible form. Expensive people still have to spend their valuable time watching the screens to spot when problems go wrong.

This problem is made even worse by the fact that the information needed by managers, their staff, and employees often exists in multiple systems, often belonging to other stove-pipe departments, where it is difficult to access.



MilramX solves these problems by continuously monitoring data, as it is generated and collected by various systems, using Artificial Intelligence to turn this data into actionable information, and then makes sure that this information gets to all the people who need it in the form of Email, Text Messages, and/or Automatically updating information in the different systems used by people in the different silos.

It should be noted that this is different from simply exchanging data between systems. People in different silos use different systems to do their jobs because these systems are tailored to the specific information needs of each silo. For example, the systems used by people doing digital marketing have a very different structure and purpose from the systems used by production or finance. Each of these systems has a database tailored to the needs of its users and collects very different data.

Simply exchanging data between system does not provide the users with the information they need. Instead, it is necessary to apply artificial intelligence to the data, which is possibly spread over multiple systems, to generate the needed information, which is the role of MilramX.

Real-Time vs. Deep Reasoning AI Systems

Real-Time AI systems like those built on the MilramX platform are fundamentally different from deep reasoning AI systems in that:

- Real-time AI systems consume and analyze data in near real-time, as the data is generated. They then provide the resultant information in real-time to assist people or control equipment. Applications in this category include self-driving vehicles and military systems.

 © Copyright Milramco LLC

- Deep reasoning AI systems, which typically use some form on non-linear matrix correlators, popularly known as "Neural Networks", start with large volumes of historical data, such as images, and attempt to draw some inference from these, such as whether someone has cancer.

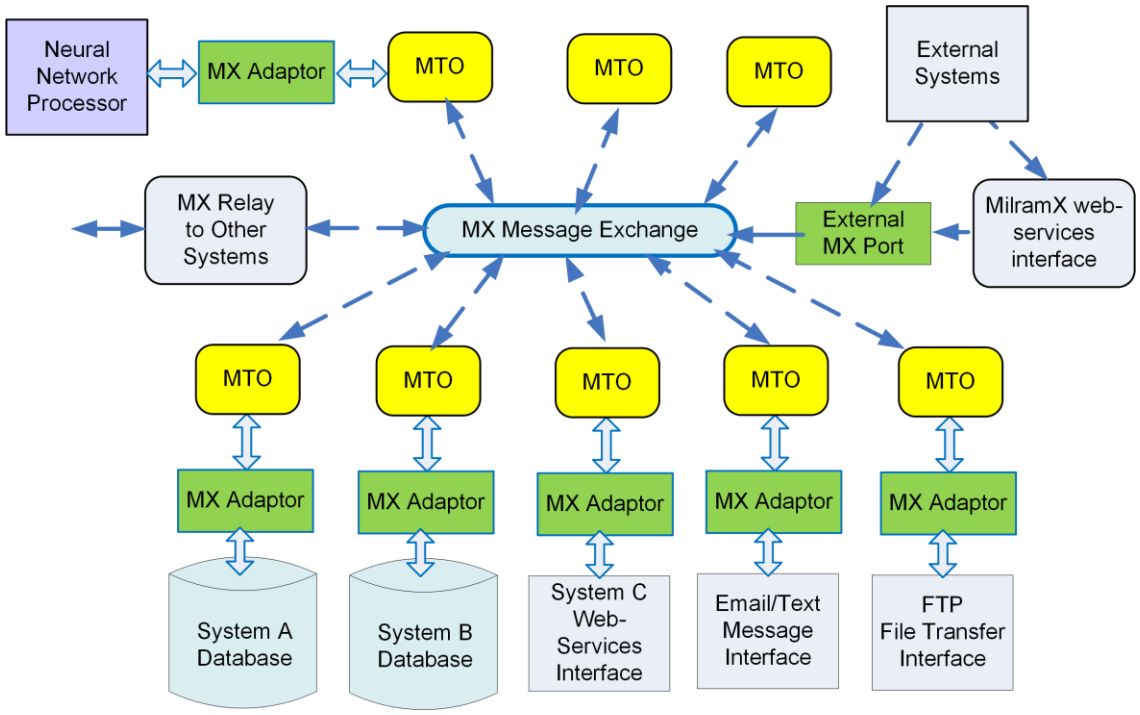
Real-Time AI systems typically combine a wide-variety of AI techniques such as decision trees, rules-based expert systems, and neural networks to integrate data from multiple sources.

MilramX integrates all these methods into a single framework using the paradigm of cooperating intelligent agents.

MilramX can use neural networks as the basis of doing deep reasoning or machine learning. However, in the case of MilramX, the data is fed incrementally to the Neural Network in real-time, along with observed outcomes, so that it can learn to make time-dependent estimates, as it is running.

How Does MilramX Work?

Overview



MilramX uses an Intelligent Agent architecture, for reasons explained in a following section. These Intelligent Agents, known as MilramX Task Objects (MTOs) exchange High Level Data Objects (HLDOs) by means of messages in order to perform their collaborative analysis of data and exchange of information.

HLDOs are named sets of name:value parameter pairs, in the manner of a Jason string. Each HLDO has a formal definition in terms of its parameters, their data type, and allowed values. HLDOs provide a convenient, computer independent, representation of data, which is human readable.

 © Copyright Milramco LLC

Some of the functions that MTOs perform are as follows:

1. Periodically monitoring tables in databases to see if there is any change to the data so that this data can be sent to other MTOs in the form of HLDOs.
2. Receiving HLDOs from other MTOs and using these to update databases.
3. Exchanging data with Cloud-based systems via their web-services interfaces.
4. Sending Emails or Text Messages to users in reponse to the receipt of HLDOs containing the Email/Text message information..
5. Sending Files to remote Servers by FTP in response to the receipt of HLDOs detailing what to send where.
6. Interfacing to processors which perform Neural Network based deep reasoning.

The messages exchanged between MTOs contain:

- Destination MTO name
- Command - what to do with the HLDO information
- HLDO - named set of name:value data pairs
- Time/Date Sent
- Importance - how important the information contained in the message is.

These messages are sent through the MX Message Exchange mechanism which forwards messages to the input queue of the designated HLDO. This mechanism also includes the ability to specify a named distribution list of MTOs, so that a single message can be sent to multiple MTOs, in the manner of an Email forwarding mechanism.

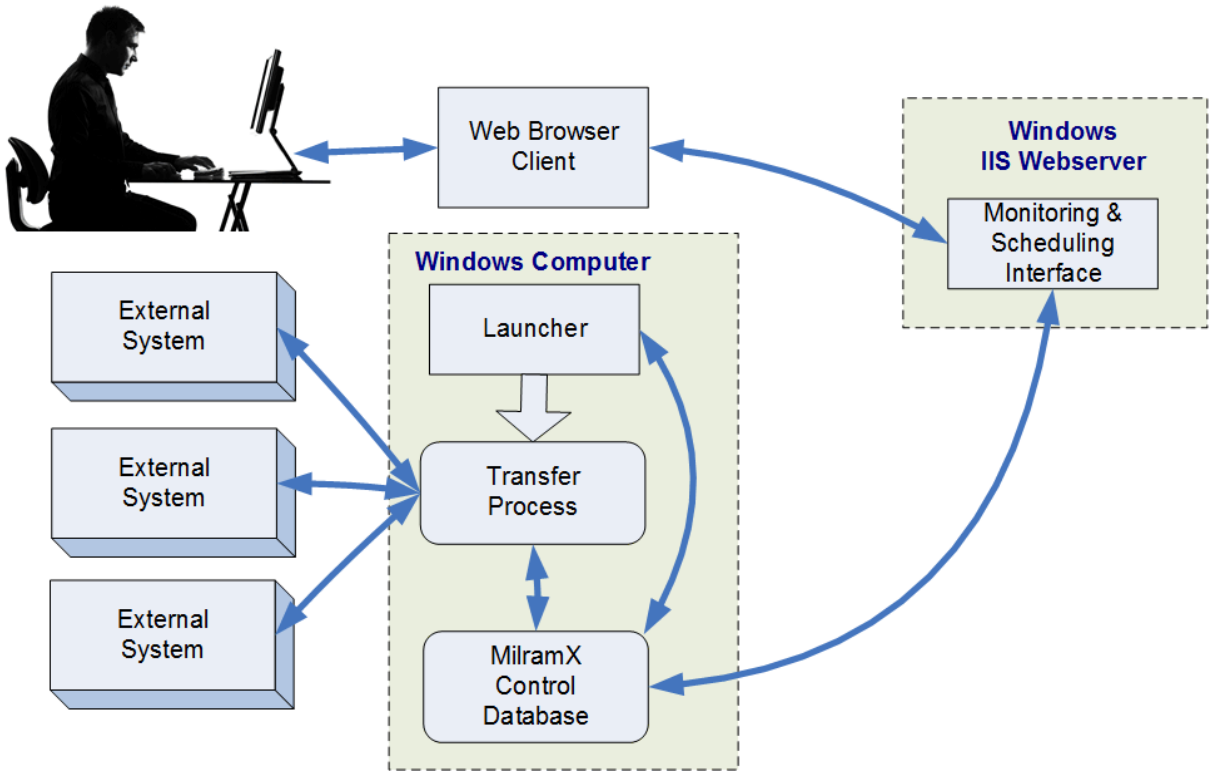
There is also an MX input port through which other systems can send HLDOs to the MX Exchange Mechanism. These messages can also be sent to this port via the MillramX web-server interface. In future it is planned to add the capability for multiple MX systems to exchange messages across processors, resulting in a true distributed real-time AI system.

The MTOs communicate with external systems via MX Adaptors, which, in the case of external databases, can use the Tau-Adaptor rules-based expert system to automate the translation of complex database structures to and from HLDOs. This enables these HLDOs to be manipulated by the MTO without needing any knowledge of the structure of an underlying database.

The MX Adaptors also perform the function of detecting bad characters in data retrieved from a source database or system, as well as validating that correctly formatted data is being sent to the target system. Where appropriate, these MX Adaptors, will also perform character set conversion, such as from Unicode to ASCII by substituting multiple ASCII characters for one Unicode character.

 © Copyright Milramco LLC

Overall MilramX Architecture



The MTOs are written as VB.Net (could be C#.Net) class objects, which act as subroutines, using the Microsoft Visual Studio .Net development environment. Here they are linked into a Transfer Process, with all the Adpator and Message Exchange mechanisms, linked as DLLs.

The Transfer Process is launched by a Launcher process, which runs as a Windows Service. This Launcher process, launces the Transfer Process at periodic times, set in the MilramX Control Database, through the MilramX web-browser based user interface.

When it starts the Transfer Process running, the Launcher specifies the name of an MTO to run. This MTO is typically one that monitors external databases or external systems for changed data. If it finds changed data, the MTO then sends this data in the form of HLDO messages to the MTOs that need to know this information. These MTOs may then send messages to other MTOs, which then send alerts to people and/or information to external systems.

The scheduler within the Transfer process will first schedule the specified MTO and, when it completes, will schedule the other MTOs, based on the sum of the importance of the messages in their input queue and the specified importance of MTOs.

The scheduler will keep scheduling MTOs until all messages are delivered, bearing in mind that MTO execution will add messages to the overall message queue. When all messages are delivered, the Transfer Process will clean-up and kill itself, so as to free up system resources for the next run of this, or another transfer process.

 © Copyright Milramco LLC

MTOs and messages use the concept of importance instead of priority. Importances are typically numbers between 1 and 10, with 1 being least important and 10 being the most important. MTOs are given an inherent importance and can set the importance of messages they send. This enables, for example, a chain of MTOs processing information about a process-line being down to take precedence over MTOs which simply log historical data.

When there are a large number of MTOs dynamically exchanging messages, based on data discovered and information generated, it is impossible to establish a priori the relative priorities (first, second, etc.) of messages and MTOs. The MilramX importance mechanism enables the scheduler to dynamically prioritize the execution of MTOs.

Some Background Information

This section was included in this data sheet, as many readers may not be familiar with real-time AI systems. Please skip this section, if you are familiar with how Real-Time AI Systems work and the history of their evolution.

What Makes Real-Time AI Systems Different?

From the dawn of the computer age, there have been two threads to the AI world:

- Real-time AI systems, which process data in near real-time, typically to advise people as to the best course of action or to take action automatically by controlling machines. Self-driving cars fall into this category, as do a variety of military systems.
- Deep reasoning analytic systems, which typically use Neural Networks to analyze large volumes of historical data. Here we see systems used for cancer diagnostics, scene analysis, as well as historical financial and sales data analysis.

Today, these are starting to merge as computers become more powerful and are able to do things like facial analysis and scene analysis in real time.

There still remains a fundamental difference, however, which is that real-time AI systems consume data in real-time, as it is generated. Deep reasoning systems start with large historical data sets and attempt to extract meaning from this historical data.

Deep reasoning systems are trained, using large historical data sets, with known relationships between the inputs (such as the pixels in an image) and the results (such as whether the image shows a cancerous skin lesion or not).

Real-time AI systems, such as MilramX, can use non-linear statistical matrix correlators (commonly called Neural Networks) as the basis of doing deep reasoning or machine learning, just like Depp Reasoning systems do. However, in the case of MilramX, the data is fed incrementally to the Neural Network in real-time, along with observed outcomes, so that it can learn to make time-dependent estimates, as it is running.

 © Copyright Milramco LLC

Neural Networks, in real-time AI systems like MilramX, are typically used to do model-based reasoning. In this, a manufacturing process may be modelled as a route of operations, with the time taken to do each operation being the dependent variable and the item being made, the quantity, the machine being used and the operator being the independent variables.

The real-time AI system then feeds the time taken for each operation along with the independent variables, to a neural network algorithm incrementally, as each operation is completed on the factory floor. From this, the system is able to learn the statistical depenency of the time for the operation on the independent variables.

We can also have the neural network learn about how long the wait is likely to be for action in each work center, based on factors, such as how many other jobs are being processed in the plant

From this we are able to predict how long each job should take, from order to shipment, based on its route and other characteristics, as well as how busy the plant is.

Why Intelligent Agents?

Artificial Intelligence, as a field, consists of a series of programming techniques to organize large amounts of computer code into systems that can advise people as to a course of action or directly take action itself, through interacting with external systems or devices.

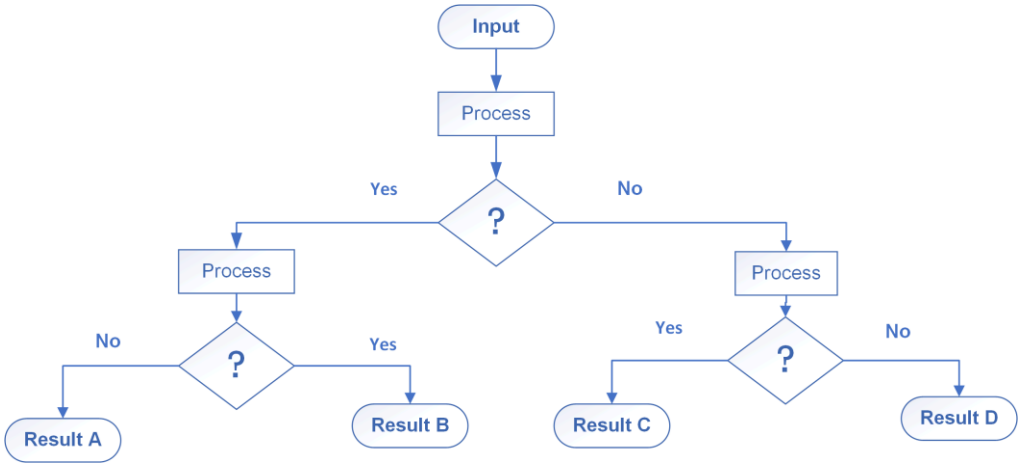
A real-time AI system comprises a number of elements:

1. Collection of data from a variety of sources in real-time.
2. Analysis of this data
3. Taking action based on the information deduced from the data in real-time.
4. Able to do collection and analysis of data from many sources in parallel.

Over the past 40 years, a number of different approaches, which are described here, have been used to develop real-time AI systems. All of these are still in use, in one form or another.

Decision Trees

Some of the earliest real-time AI systems were based on decision trees, where inputs to the system were used to traverse a fixed tree with conditional rules applied at each level of the tree.



 © Copyright Milramco LLC

These are still used where the size of the decision tree is limited, the outcome of each decision process can be expressed in a yes/no decision, and the decision flow can be hard coded.

Applications today include ladder-logic programming of PLCs (Programmable Logic Controllers) as well as SCADA (Supervisory Control and Data Acquisition) systems used in process control and some algorithms within self-guided cars.

This paradigm breaks down, as the code becomes impossible to manage, debug or easily modify when:

1. The Decision Tree becomes too large to handle in one body of code - too many decision processes and too many branches with too many levels.
2. The output of each decision process is not a yes/no but a set of answers with different probabilities. This occurs where the decision process is some form of statistical correlator, such as a neural network.
3. The decision flow cannot be expressed in a hard-coded tree hierarchy. Here data needs to flow between different decision processes depending on the outcome of the analysis, in the form of a connected graph, rather than a tree.

These days, building an AI system is an iterative process, which requires adding and changing decision rules and processes in an "Agile" software development process. Except in simple cases, decision trees do not lend themselves to this programming style and are much more effective with the older "Waterfall" development methodology where all the decisions are hard-coded based on a fixed specification.

Expert Systems

Expert Systems were invented to overcome some of these problems.

At the core of an Expert System is a set of "IF condition THEN change some data item" rules. The preconditions are dependent on a set of data items, which may be the input variables to the set of rules, a set of internal variables, which can be set by the rules, and a set of output variables, which can also be set by the rules.

The preconditions for the rules can be based on the values any of the input, internal, and external variables, such as "IF sales_inquiry.state = NY THEN assigned_sales_person = Fred". Initially some rules are triggered by the input data variables. These rules may generate changes to intermediate and output variables, which trigger more rules.

This process continues until there are no more rules to fire, when the output variables are used, by the program in which the expert system is embedded, to take some action, such as sending Fred an Email about his sales leads.

A big advantage of rules, when using Agile development, is that new rules can be added incrementally without modifying existing rules. Also, rules are very readable by non-programmers, provided suitable variable names are used, making it easy for operations, sales,

 © Copyright Milramco LLC

and marketing staff to verify visually that each rule is correct, as shown in the example shown below:

```
1     RCO temperature_control is -- Name of the RCO is "temperature_control
2     declare
3         temperature is FLOAT;  -- The temperature is a floating point variable
4         heater is (ON, OFF);  -- Heater is defined as a symbolic data type
5         control is (RUN, STOP); -- Triggers the RCO with RUN, STOP commands
6         receive temperature, control; -- Attaches RCO to temp. and control DDNs.
7         autosend heater;  -- Heater sent out as a message to the heater DDN.
8     init  -- assumed initial values
9         pragma period is 500 msecs;
10        temperature := 60;
11        heater := OFF;
12    begin  -- Start of RCO rules
13        -- The RCO only wants to send out a heater message if the controller
14        -- has not been modified within the last three seconds
15        if temperature > 100 and heater = ON and tnow - heater'time > 3 seconds
16              and control = RUN then heater := OFF;
17        if temperature < 90 and heater = OFF and tnow - heater'time > 3 seconds
18              and control = RUN then heater := ON;
19        if control = stop and heater = ON then heater :=OFF;
20    end;
```

The problem with expert systems is that not all decision making can be expressed in IF..THEN.. rules and often involves complex mathematical calculations. As a result, the rules have to be able to call subroutines written in an algorithmic language to do the calculations.

A big issue with Expert Systems is debugging. As soon as you have more than half a dozen rules or so it becomes very difficult to follow the flow of decision making, as this is not hard coded, and thus amenable to conventional software debugging methods. Also, if the rules call subroutines written in a procedural language, to do complex calculations, the debugging becomes even more complicated

One place where rules-based techniques remain in use is where the IF THEN ELSE rules are embedded in well debugged code but business analysts can change the parameters of the rules to tailor the behavior of the rules to the needs of the individual organization.

This is done in the BellHawk work-in-process and materials-tracking software, where the parameters can be set for rules that:

- Collect users defined parameters
- Issue warnings when users are about to make mistakes
- Generate barcode labels with custom data fields

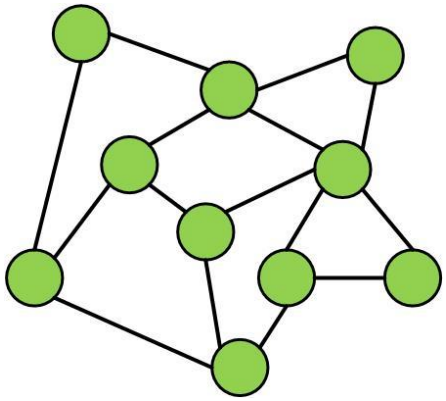
The parameters for these are set using Excel imports, thus avoiding the need for business analysts to do any software coding.

This same approach is used within MilramX itself where the Tau-Adaptor rules-based expert system is used by the MX Adaptors to automatically convert between Fetch, Store, and Lookup requests, issued by an MTO, and the complex structure of most operational databases.

 © Copyright Milramco LLC

Intelligent Agents

The next evolution in real-time AI systems is the paradigm of intelligent agents, communicating by messages. This was pioneered by the Activation Framework project, which was developed by a team led by the author in the 1990s with funding by the USAF.



In Activation Framework, the intelligent agents were coded using a special Expert Systems language, called Decision Support Language. Unfortunately, as soon as the number of rules in an intelligent agent grew to be more than about half-a-dozen, it became very difficult to debug the intelligent agent code. Also, we found that much of the code needed within each intelligent agent did not fit well within the IF THEN ELSE coding paradigm but looked much more like the code used in decision trees.

Other than that, the paradigm worked well for building large complex decision-making system, as we could, with some difficulty, debug the rules each intelligent agent separately. Or we could write them directly in the C language, which made debugging much easier. Then we could "knit" these agents together in a flow-graph linked by messages, with agents being triggered by the arrival of messages, in a manner analogous to the triggering of rules in an Expert System.

This paradigm of intelligent agents cooperating by sending messages, was then used for the development of the MilramX real-time AI software platform. This platform has been extensively used to implement systems which exchange information between industrial systems. In its early development this was especially between the BellHawk real-time work-in-process and materials tracking system and a variety of accounting and ERP systems.

MilramX is now used in industrial alerting systems, such as warning material handlers when they need to replenish inventory in Kanban bins on the factory floor. In addition, it is increasingly being used to exchange information for supply chain traceability in food and pharmaceutical applications.

The biggest weakness of the collaborating agent paradigm is that, in practice, we develop, test, and debug each intelligent agent as an individual entity and then we put them all together in a system, which is then tested operationally, as whole. This is the same methodology used to build and test complex systems such as aircraft. It suffers from the same flaws in that unexpected interactions between the "black box" agents can cause unexpected failures. Fortunately, in information delivery systems, this is usually not as fatal as a systems interaction flaw in an aircraft.

Another weakness is that the agent code has to be coded by programmers. This code can be simplified through the use of high-level calls such as Fetch, Store, Send, and Receive that make it possible for business analysts to understand the code. It has, however, also proven possible for programmers to write agent code that is completely incomprehensible to non-programmers (as with any other system development methodology).

 © Copyright Milramco LLC

MilramX Task Objects

MilramX uses the same communicating-intelligent-agents paradigm as Activation Framework but each agent is written in regular VB.Net (or C#.Net) code which allows for a wide-range of analytic methods to be used with each agent. This enables the code in each agent to be easily debugged, using conventional step-by-step tracing methods, but then enables agents to be added incrementally into an overall system, as the system evolves.

One thing that we could have lost in this process was the ease with which business analysts could easily read the code for each agent. The other issue was the ability to easily generate code to make decisions without having to generate a lot of code to "glue" the decision code to the rest of the system.

To this end, we did the following with MilramX:

1. Made HLDOs the standard data object paradigm so as to simplify data manipulation such that MTOs did not have to be concerned with the source system for the data they received and the formats which that system used for data.
2. Reduce the number of standard subroutine calls, through which an MTO interacts with the rest of the system (or external systems) to
    a. **Send** a message containing an HLDO to another MTO or set of MTOs.
    b. **Receive** a message containing an HLDO from the MTO's input queue.
    c. **Lookup** HLDO data based on HLDO parameter values through an MX Adaptor.
    d. **Store** an HLDO in a database or send it to a remote system.
    e. **Fetch** an HLDO based on the latest changes to a database or system.
    f. **Save** an HLDO in the MTOs permanent store.
    g. **Retrieve** an HLDO from the MTO's permanent store.
3. Provided a set of functions to simplify HLDO manipulation, such as retrieving or storing an individual parameter value in the HLDO by parameter name.

As a result, while it is possible to write very complicated MTO code, most MTOs can be coded in a simple and easily understood way.

These MTOs are typically coded in VB.net using Microsoft Visual studio and then linked with Dynamic Link Libraries, which are provided with MilramX and provide over 90% of the needed application code, into a Transfer Process. This Transfer Process is then loaded onto the target computer, where its execution is controlled by the Launcher process, as described previously.

 © Copyright Milramco LLC

Commentary

MilramX is proving to be an excellent platform with which to implement real-time information exchange systems within an industrial enterprise. This is because it runs on Windows Workstations and Servers and is compatible with databases and software found in many industrial enterprises in the USA.

Up to now, MilramX has been primarily used to exchange information between the BellHawk operations and materials tracking system and a variety of other systems, such as ERP and accounting systems. It has also been used to distribute advanced shipment notice data in the global supply chain, as well as being used for alerting people, such as when shop-floor inventories run low.

We are now expanding the application of MilramX to include CRM and Marketing Automation systems, so as to enable much closer information coupling between sales, marketing, and the production and materials management departments within industrial organizations. In addition, we are working on closer coupling with process control systems using Industrial Internet of Things (IIOT) processors on the shop floor.

MilramX is, of course, simply a software platform, which makes implementing these systems much easier by:

1. Providing an architectural framework that embodies best practices in implementing this class of system.
2. Providing over 90% of the code needed for implementing these systems, pre-built or automatically generated.

These, taken together, significantly reduce the time and cost for implementing these systems and also dramatically reduces the probability of implementation failure.

Having said that, however, it is critically important to work with a team of people who have experience in implementing this type of system from initial information gathering about the goals of the organization, through coding of the MTOs, to deployment training and support.

KnarrTek can provide people with in-depth experience in these methods, and in using MilramX, to work with your team in implementing a system which can use AI to significantly increase your sales, through improved customer interaction, while dramatically cutting costs by reducing the number of overhead support people needed to find prospects, close sales, and deliver products on time.

 © Copyright Milramco LLC