Using Grafcet to Design Generic Controllers

Bruce Hunter Thomas and Charles McLean
Factory Automation Systems Division
National Bureau of Standards
Gaithersburg, MD 20899

Abstract:

The purpose of this paper is to outline some ways Grafcet is being used to help design generic controllers within the Automated Manufacturing Research Facility. Grafcet is a powerful graphical language for expressing control flow and allows the expression of both parallel and sequential control logic. The paper describes the rationale for using Grafcet as a design tool for expressing software for a generic controller. Grafcet helps the designer determine: modularization of the code, functions that can be performed in parallel, communication between parallel processes, and problems in control flow.

1. Introduction:

The cost of developing software for the factory is continuing to increase. It is a well known software engineering practice to develop reusable software whenever it is feasible [1]. There obviously needs to be a consistent philosophy for developing reusable code for manufacturing controllers systems. There have been some efforts in the past to develop what is termed as a "generic controller" [2],[3]. We have chosen Grafcet* as the design environment to continue research into the generic controller.

Grafcet is a powerful graphical language for expressing control flow; it supports the expression of both parallel and sequential control logic. Grafcet is based on techniques developed in the Petri net language[4],[5] and is an excellent tool for documentation and demonstrating the control flow of a manufacturing control system. This papers discusses the use of Grafcet as a design and documentation tool for building a generic control system. Examples of what makes Grafcet a good design environment are described herein.

For a better understanding of our approach to generic control, a definition of a generic controller and an overview of the Grafcet language and the Automated Manufacturing Research Facility (AMRF) facility is presented. An argument is presented for using Grafcet to design manufacturing controllers. Methods for using Grafcet as a design tool are shown through the use of examples. A discussion is given of translating this design into a procedural language.

2. Background:

2.1 AMRF Overview

A major goal of the AMRF project has been the establishment of a testbed small batch manufacturing system at the NBS Gaithersburg, Maryland site. The testbed, which became operational in 1983, is designed to be used by government, industry and academic researchers for the development, testing and evaluation of potential interface standards for manufacturing systems [6]. The NBS testbed differs from virtually all other flexible manufacturing systems in the variety of "off-the-shelf" components that have been integrated into a single coordinated operation. To ensure that as many critical system integration issues as possible are addressed, component modules are chosen from many different systems vendors. Integration test runs of the AMRF have demonstrated the automated production of small batches of machined parts.

The results of AMRF research are being transferred to the private sector via the NBS Industrial Research Associate Program, participation in factory standards organizations, conference papers, technical reports, and various technology transfer programs that have been established with project sponsors and other participants.

2.1.1 Testbed Description

The industrial machinery of the AMRF occupies a 5000 square foot area of the NBS machine shop. The factory systems that are located on the floor of the AMRF includes: two machining centers and a turning center, a coordinate measuring machine, six robot manipulators, a vision system, two wire-guided vehicles, storage and retrieval systems, tray roller tables, tool setting stations, vacuuming and other cleaning equipment, part fixturing and robot gripper systems.

---

* The Grafcet used in this paper is the language defined by Savoir, Oakland, CA, which has some very useful extensions. Savoir provides an interactive development environment FLEXIS DESIGNER™ to help design control systems with Grafcet. There is a research agreement between NBS, Savoir, and Xerox Corporation for the use of FLEXIS DESIGNER™ and two Xerox 1186 workstations. The FLEXIS DESIGNER™ runs on Xerox 1186 artificial intelligence development workstation.

---

For a better understanding of our approach to generic control, a definition of a generic controller and an overview of the Grafcet language and the Automated Manufacturing Research Facility (AMRF) facility is presented. An argument is presented for using Grafcet to design manufacturing controllers. Methods for using Grafcet as a design tool are shown through the use of examples. A discussion is given of translating this design into a procedural language.

2. Background:

2.1 AMRF Overview

A major goal of the AMRF project has been the establishment of a testbed small batch manufacturing system at the NBS Gaithersburg, Maryland site. The testbed, which became operational in 1983, is designed to be used by government, industry and academic researchers for the development, testing and evaluation of potential interface standards for manufacturing systems [6]. The NBS testbed differs from virtually all other flexible manufacturing systems in the variety of "off-the-shelf" components that have been integrated into a single coordinated operation. To ensure that as many critical system integration issues as possible are addressed, component modules are chosen from many different systems vendors. Integration test runs of the AMRF have demonstrated the automated production of small batches of machined parts.

The results of AMRF research are being transferred to the private sector via the NBS Industrial Research Associate Program, participation in factory standards organizations, conference papers, technical reports, and various technology transfer programs that have been established with project sponsors and other participants.

2.1.1 Testbed Description

The industrial machinery of the AMRF occupies a 5000 square foot area of the NBS machine shop. The factory systems that are located on the floor of the AMRF includes: two machining centers and a turning center, a coordinate measuring machine, six robot manipulators, a vision system, two wire-guided vehicles, storage and retrieval systems, tray roller tables, tool setting stations, vacuuming and other cleaning equipment, part fixturing and robot gripper systems.
The shop floor area is partitioned into six manufacturing workstations:
- Horizontal Machining
- Vertical Machining
- Turning
- Cleaning and Deburring
- Inspection
- Material Handling

A typical workstation is built around a numerically controlled machine tool or some other major piece of equipment that is central to the station's role in the facility. It is also usually consists of an industrial robot, one or more control computers, local material storage areas, sensor systems.

Outside of the shop floor area, four control rooms house computer terminals and other peripherals that provide user and research interfaces for control, data administration and communications systems. In a nearby manufacturing engineering area, a number of graphics devices, terminals, computers and a plotter are located to support the following functions:
- part design and modeling,
- group technology (GT) classification and coding,
- process planning, and
- offline control programming.

Additional supporting equipment are located in laboratories and offices in adjoining buildings.

2.1.2 System Architecture

A system architecture has been developed which integrates the subsystems of the AMRF small batch manufacturing system. The workstations, which have been integrated, were constructed from components obtained from many different systems vendors. The integration software that holds the AMRF system together was developed primarily at NBS. The key architectural concepts that facilitated integration included:

1) the decomposition of the manufacturing management hierarchy into well-defined levels,
2) the uniform treatment of work order decomposition across the hierarchy,
3) the definition of standard work elements for all control systems,
4) a consistent data structure for process plans at all levels in the control hierarchy, and
5) uniform interfaces to system services, such as: network communications and data administration.

2.2 Generic Controller

AMRF research efforts have focused on the identification of generic control principles which could serve as a basis for the development of future interface standards for factory control systems. By recognizing generic functional requirements for control systems, it is hoped that redundant software development efforts can be minimized. For example, all controllers within the AMRF must support a common basic set of functions:

1) the processing of control messages expressed in the AMRF command/status message format,
2) the exchange of data with the AMRF common databases through Integrated Manufacturing Data Administration System (IMDAS),
3) the retrieval, interpretation and execution of process plans that have been prepared in a standard AMRF process plan format,
4) the transition of initialization and shutdown states according to a protocol developed for the AMRF by the University of Virginia,
5) the transmission of message through AMRF common memory areas and communications networks, and
6) the input and display of controller data through a human interface system.

It will now be illustrated how Grafcet can be used to develop a generic manufacturing controller that embodies many of these concepts.

The current AMRF cell control system is a software package developed at NBS which coordinates and integrates the activities of subordinate workstation level control systems [2]. The cell software is implemented entirely in Microsoft™ C and executes on IBM™ PC and compatible computers under the MS-DOS/PC-DOS™ operating systems. A common memory system and serial communications server implemented on a Sun™ computer workstation provides a communications link between the PC-based controller and the AMRF communications network. Communications interfaces to the cell control computer running on the Sun are also implemented in the C programming language.

The generic software modules which make up the cell controller include:
- System Supervisor (SS) -- which sequences the activation of major subsystems,
- Communications Manager (CM) -- which handles the sending and receipt of message traffic,
- Data Manager (DM) -- which handles the translation of data to/from internal formats from/to external message formats,
- Manufacturing Manager (MM) -- which decomposes commands, process plans, and sequences work order processing,
- Transition Manager (TM) -- which sequences the system through major evolutions, such as: initialization, restart, error recovery and shutdown,
- User Interface Manager (UIM) -- which handles the user interface.

Other components of the cell include: 1) the library of functions or subroutines, called work elements, that provides basic manufacturing capabilities, 2) an internal world data model, 3) diagnostic and test software, and 4) support software which provides access to data files, builds new versions of the system, etc.
The SS is responsible for initializing the cell system, activating each of the other major system modules during normal operations, and coordinating orderly system shutdowns, and . The System Supervisor activates each module at least once per control cycle. The system, running on a COMPAQ Deskpro 386TM, executes about 10 control cycles per second.

At the beginning of each control cycle, the SS advances the system clock and updates counters and timers, (Figure 1.) Next, the SS invokes the UIM to read the keyboard and update the screen. The UIM is called before each control module to accept user input and update monitor output. The CM is then invoked in RECEIVE mode, giving it the opportunity to handle incoming communications. Following this, the DM is activated in INPUT mode translating newly received messages into internal data formats. The TM is invoked to update its state table and sends transition commands to the subordinate controllers as needed. The MM is activated to decompose new work orders or continue the sequencing of old work orders based on new status information. New control commands and status data are written into the internal common memory areas. Requests for external data, such as process plans, are also entered. After the MM completes its processing, the SS activates the DM in OUTPUT mode. The DM checks internal data structures for changes and generates messages in prescribed external formats, as required. Finally, the SS invokes the CM in SEND mode. The CM checks to see if it is time to transmit messages and then sends new outgoing messages across a serial link to a SunTM computer which provides a common memory interface to the AMRF network. The SS loops and continues to repeat the entire sequence until shutdown.

2.3. Grafcet Terminology

To help in the understanding of Grafcet and how the language is being used in this paper, an overview of the language is given here. Grafcet is a powerful graphical language for expressing control flow. Savoir has implemented a version of Grafcet which is being used to design the generic controller[7]. Grafcet is an excellent tool for, designing, documenting and demonstrating the control flow of a system. The basic primitives of the Grafcet language are described in detail, and the parallel control constructs in Grafcet are explained. Savoir has augmented Grafcet and these extensions are outlined here.

2.3.1. Primitives

The Grafcet language has three major primitives: regular steps, macro steps, and transitions. Regular steps allow the user to represent an arbitrary control action in the form of embedded C source code. Macro steps allow the user to name embedded Grafcet macro programs as control actions. Transitions act as gates on the flow of control through the Grafcet program. Associated with each transition is a C expression which determines if control can pass through the transition. If the expression evaluates to False, the gate stops the control flow. If the expression is True, the gate allows the control flow to continue. These primitives can be connected to form a control flow path by attaching a link between two primitives. A link can only be attached between a step primitive and a transition primitive; that is to say a transition primitive cannot be linked to another transition primitive. These links are represented by a line from one primitive to another; sometimes an arrowhead is used to indicate the direction of the control flow. Examples of what these primitives look like are shown in Figure 2. In these examples 0 and M1 are labels supplied by the Grafcet programming environment, and label is a user supplied name for a step.

2.3.2. Synchronization

Grafcet allows a single step to be followed by more than one transition, and a single transition to be followed by more than one step. The terms "fan-in" and "fan-out" are sometimes applied to this idea of multiple links. The two types of fan-outs are synchronous and asynchronous branches and fan-ins are synchronous and asynchronous joins.

A synchronous branch is when a transition is followed by two or more steps. When the transition is true, all of the steps are initiated simultaneously. A synchronous join is when a transition is preceded by two or more steps. Control cannot proceed after a synchronous join until all the preceding steps have completed and the single transition becomes true. An example of a synchronous branch and a synchronous join is shown in Figure 3. The double horizontal lines in the figure indicate the synchronous branch and join.

An asynchronous branch is a single step followed by two or more transitions. An asynchronous join is two or more transitions followed by a single step. Control can flow through any of the transitions whose conditions are true. Because of the nature of the asynchronous branch, multiple tokens can be produced from an asynchronous join. Tokens are used to denote the control flow through the Grafcet program. This can be a problem when translating this design to standard procedural language. An example of an asynchronous branch and an asynchronous join is shown in Figure 4. The single horizontal lines in the figure indicate the asynchronous branch and join.

2.3.3. Savoir enhancements

These are some of the features which Savoir has augmented in the Grafcet language to make the language more powerful and easier to use. The Grafcet transitions use C expressions for the condition that determines whether or not control can pass through the transition. In an asynchronous branch the special transition condition of otherwise was added. Otherwise is defined as a condition that is true if the conditions of the transitions to its left in a set of asynchronous branches are false. Savoir augmented Grafcet
to support C source code in their "regular step."

3. Why use Grafcet?

What makes Grafcet a useful tool for designing manufacturing controllers? Grafcet uses a graphical notation for its language. The graphical representation of control makes for easy understanding, documenting, and prototyping. Grafcet also has parallel constructs inherent in the language. These constructs allow the user to represent distributive systems in an elegant manner. Finally, Savoir has enhanced Grafcet with a user friendly programming environment.

3.1. Parallel nature of Grafcet

A design decision was made to implement the generic controller as a set of parallel processes. The existing cell controller in the AMRF was designed and built using the concepts of a generic controller, but its implementation was done as one serial process. A model of the cell controller working in the AMRF factory is outlined in Grafcet (see Figure 5). The new generic controller will be built on the model of the existing cell controller. This new controller will separate the generic components into distributive processes. A major purpose of this design is to define the parallel portions of this generic controller and the interfaces between the parallel portions of the controller. Grafcet is an excellent means for designing distributive systems; because Grafcet allows for both synchronous and asynchronous processes to be simulated on one machine.

3.2. Quick prototyping

The Savoir programming environment includes a graphical editor, a MAP network interface, and an interactive interpreter. With the use of the graphical editor in this Grafcet system, a controller can be modeled quickly. The menu-based graphical system takes its input from the mouse and keyboard.

The programming environment allows the user to execute a design interactively. The control flow is denoted as tokens flowing through the Grafcet program. An incremental compiler for both the Grafcet program and the C source code is included in the environment to allow for timely executions of the design. This allows the designer to test out ideas while they are still designing the system. Grafcet programs can be executed and the control flow monitored as tokens pass through a graphical representation of the Grafcet program. This interactive approach to design allows the designer to look for problems dynamically, such as deadlock or race conditions. Since a system design becomes a working model, a translation step from design to first implementation is not needed. There are interactive debugging tools such as: single stepping, display of state variables, and selecting which modules are to be currently displayed.

3.3. Easy to visualize control flow

The control flow of a controller designed in Grafcet is easy to visualize because of the graphical nature of the language. The language is powerful in its ability to represent real-time distributed systems. Grafcet provides this functionality with a small set of programming constructs. With the programming environment, a design can be represented as a dynamic entity. Once the design is finished, it can also be used as a functional model of the working system.

3.4. Good documentation

Because of its graphical nature, Grafcet is a useful documentation tool. Since the design is in a graphical form, there is no need to translate the design into a form which is easier to read. The nature of Grafcet forces one to do a top down design, which provides many levels of detail for the design. This top down approach helps to organize documentation for a control system.

4. Grafcet as design tool

Some examples on how Grafcet helps in designing a controller are now shown. These examples have been simplified to help clarify the concepts of designing with Grafcet. Grafcet allows for parallel execution of program modules; therefore Grafcet can be used to determine which functions in a controller can be executed in parallel. Grafcet allows for both synchronous and asynchronous control, and these control constructs are examined in more detail. The design of interprocess communication is also discussed.

4.1. Synchronized control

The generic controller has five managers (software modules): mfg_mgr, trans_mgr, data_mgr, comm_mgr, and ui_mgr. In the Grafcet program shown in Figure 5, mfg_mgr, trans_mgr, data_mgr, comm_mgr, and ui_mgr, operate in parallel. Each of these managers is placed in a macro step, which is synchronized with the other manager macros. The system will not enter exit_code until all of the managers have finished processing and program_exit is True. The managers finish processing when the controller is in shutdown mode. The system then passes control to start the cell again in init_code. This Grafcet program helps to determine the control functions of each of the managers by keeping the control flow of each manager completely separate.

4.2. Asynchronous control

The idea of asynchronous control is very clearly shown in the comm_mgr. The comm_mgr handles the communication between a supervisor controller and its subordinate controllers. The subordinate controllers communicate varying amounts of information with the supervisor at independent intervals. An example in the AMRF is the cell controller. The cell controller is the supervisor of five workstation level controllers: Material Handling Workstation (MHS), Vertical Workstation (VWS), Horizontal Workstation (HWS), Inspection Workstation (IWS), and Cleaning & Deburring Workstation (CDWS).
An example of how these five workstations could communicate in an asynchronous manner is shown in Figure 6. In this example MH5, VWS, HWS, IWS, and CDWS are modules monitoring the communication with each of the five subordinates. An example of how this communication scheme would look in a synchronous manner is shown in Figure 7.

4.3. Determining parallelism

One problem in controller design is determining which functions in a software module can be processed in parallel. The Data Manager in the generic controller is an example of this problem. The Data Manager acts as a buffer between information outside the controller and the data model inside the controller. There are three main functions of the Data Manager: decode incoming information, encode outgoing information, and interface with the database system. A serial implementation of the Data Manager is illustrated in Figure 8. The modules, InputData, OutputData, and DatabaseData are activated serially in a loop until the controller is in an exit condition. The next possible step to determine if these modules could run in parallel is to place them in a case statement (see Figure 9). By placing the modules in a case statement, the execution order is now not predetermined. The last step is to place these modules into asynchronous branch and join (see Figure 10). If these modules can run in an asynchronous mode, parallel task constructs, such as those found in Ada[8] or UNIX[9], could be used to implement these modules.

4.4. Use of Actors

A major design issue in the generic controller is the interprocess communication between the five managers. Grafcet provides a mechanism for communicating between parallel processes linked over a network. Grafcet has the notion of an actor. An actor is an autonomous process having a few well-defined interdependencies with other actors. The communication between actors is handled with a MAP network using the MMFS format.

The power of using an actor is that information passing is forced through the network. This is useful in determining how information should be passed between parallel processes. A comparison between shared data and peer to peer communication can be made using actors. One actor can be used to control all the managers in a shared data example, or a single actor used for each distributive generic manager in a peer to peer communication example. An example of the top level of a transition manager actor which controls peer to peer communication scheme is shown in Figure 11.

5. Translating a design in Grafcet into a standard procedural language

There are two main concerns pertaining to translation of a Grafcet design into an implementation in a procedural language: parallelism and data representation. Grafcet is a more powerful language than most procedural languages in its representation of control flow. It can be thought of as a superset of procedural languages[10]. The parallel execution of modules is not inherent in the more common programming languages: Fortran, Pascal, C, Basic, or Cobol. There are exceptions: Ada, Modula II, and others. The expression of local parallel modules, as in the example of the communication manager, is difficult in these parallel languages. In the case of a language with no mechanism for parallel execution of software, the operating system must supply the needed parallelism. The power in using Grafcet as a design tool is that Grafcet helps determine the most reasonable parallel design. A design can be tested before beginning the difficult and expensive process of implementation using the standard software tools that are available today.

In the case of data representation, Grafcet is a subset of procedural languages. It is difficult to design the data flow in Grafcet. All of data flow design is performed in the C source code. Therefore, there is a risk of developing a poor data flow model due to the lack of data modeling tools in the Grafcet language.

These problems are mentioned not as a criticism of Grafcet, but to make the potential user aware that some care must be taken in using Grafcet as a design tool. As in all tools, this is not "the environment to end all environments." It is a giant leap forward in the area of user friendly programming environments. The idea of building parallel control constructs which are straightforward, easy-to-use, and embedded into the language certainly is a major positive feature. Building network code right into the language is also a novel and powerful idea. These are some of the reasons Grafcet was chosen for a design environment for the generic controller.

6. Conclusion

Grafcet has proven to be a useful tool for designing a generic controller. The graphical nature of the language makes Grafcet easy to learn and use. Therefore the entire design team was able to start using Grafcet in a timely manner. The ability to test different ideas quickly has been very useful in determining the final design. The features in Savoir Grafcet programming environment played a major role in making this a useful design tool.

There are some drawbacks to the programming environment. Macros can only be used once after they have been defined. Also, Grafcet does not support any data representation in the language. The data flow in a Grafcet design has to be modeled in C source code. Therefore, a system's data flow is not very obvious to an end user looking at a Grafcet design.

There are a few problems with translating a Grafcet design into procedural language system implementation. A system can be elegantly designed in Grafcet, but it could be very
difficult to implement in a procedural language. The lack of data representation in the Grafcet language creates some problems from a software engineering view point. There are no parameter passing facilities in the language. All data passed in a program is done by global variables.

Grafcet itself can be used as an implementation language. Thus developers do not have to translate from a design environment to an implementation environment.

Overall, Grafcet is an extremely useful tool for designing manufacturing controllers.

References:

The NBS Automated Manufacturing Facility is partially supported by the Navy Manufacturing Technology Program.

Certain commercial equipment, instruments, or materials are identified in this paper in order to adequately specify the experimental procedure. Such identification does not imply recommendation or endorsement by the National Bureau of Standards, nor does it imply that the materials or equipment identified are necessarily the best available for the purpose.

This is to certify that the article written above was prepared by United States Government employees as part of their official duties and is therefore a work of the U. S. Government and not subject to copyright

---

```
comm_mgr  -  Receive messages from Network

data_mgr  -  Send database request and decode messages

trans_mgr -  Update transitions

mfg_mgr   -  Perform application functions

data_mgr  -  Receive database reports and encode messages

comm_mgr  -  Post messages to Network
```

Figure 1  Cell Controller
Figure 2  Grafet Primitives

Figure 3  Synchronous Branch and Join

Figure 4  Asynchronous Branch and Join

Figure 5  Generic Controller
Figure 6  Asynchronous Communication

Figure 7  Synchronous Communication
Figure 8  Serial Data Manager

Figure 9  Case Statement Data Manager
Figure 10  Parallel Data Manager

Figure 11  Example of an Actor