IRTC: Integration of Knowledge Engineering with Real-Time Control

Tapio Taipale Douglas Foxvog

VTT Electronics, FIN-90571 Oulu, Finland

Abstract

Integration of knowledge engineering into a control system introduces problems. They are caused by the fact that timing information for a knowledge engineering task is usually not well known. The indeterminate timing causes problems for real-time scheduling. In this paper a new approach is presented to solve the problem. The approach includes, among other things, usage of multiple versions for tasks, selection of versions dynamically according to the available time, and classifying of tasks into mandatory and optional tasks to ensure proper operation for critical operations. The approach is result of the ongoing EUREKA project called IRTC.

1. Introduction

Nowadays, machines and devices are becoming increasingly sophisticated. Machines are being equipped with intelligent operations, and reliability is increased by including diagnostics in the control systems. A common way to implement these features is to use knowledge processing. However, most control systems operate under real-time requirements. Integrating knowledge processing with real-time control leads to conflicts. Rule-based expert systems traditionally operate significantly slower than algorithmic routines and generally take an indeterminate amount of time to reach an answer, depending upon the amount and value of data in the system. Contrarily, real-time systems require guaranteed completion of routines within fixed, generally short, time periods. These contradictions make the marriage of the two technologies difficult.

Several methods have been introduced to solve the problem of integrating real-time and AI, such as anytime algorithms [1], approximate processing, imprecise computing, and progressively deepening reasoning. However, all those methods concentrate on implementing a knowledge processing system with real-time characteristics, instead of a real-time system with knowledge processing features. When, for example, diagnostic features are added to a control system, the main goal is to add appropriate knowledge processing to the real-time control system -- not to create a knowledge engineering system.

In the EUREKA project entitled Intelligent Real-Time Control (IRTC, no. EU711), we are working on an integrated system that enables development and execution of intelligent control systems with real-time characteristics [2]. The main benefit of our approach is that while the control system contains well-behaved tasks with real-time requirements, it can also include reasoning tasks and other tasks for which the execution time is difficult to determine beforehand. The running of an IRTC system is designed so that all the mandatory tasks are always performed within their given deadlines while the optional tasks are executed only if there is enough time for them.

This paper concentrates on describing those aspects of IRTC which relate to question of integrating AI and real-time. First, the general architecture of an IRTC system is outlined. Then, a more detailed description of running of a IRTC application is given, putting emphasis on AI and real-time aspects. Finally, a conclusion is presented. For the other features of IRTC, see [3].

2. Architecture of IRTC system

2.1. Application structure

An IRTC application consists of application states, activities, and activity versions (Figure 1). The system is always in one of the application states as determined by the contents of the application state vector.

1068-3070/94 $03.00 © 1994 IEEE

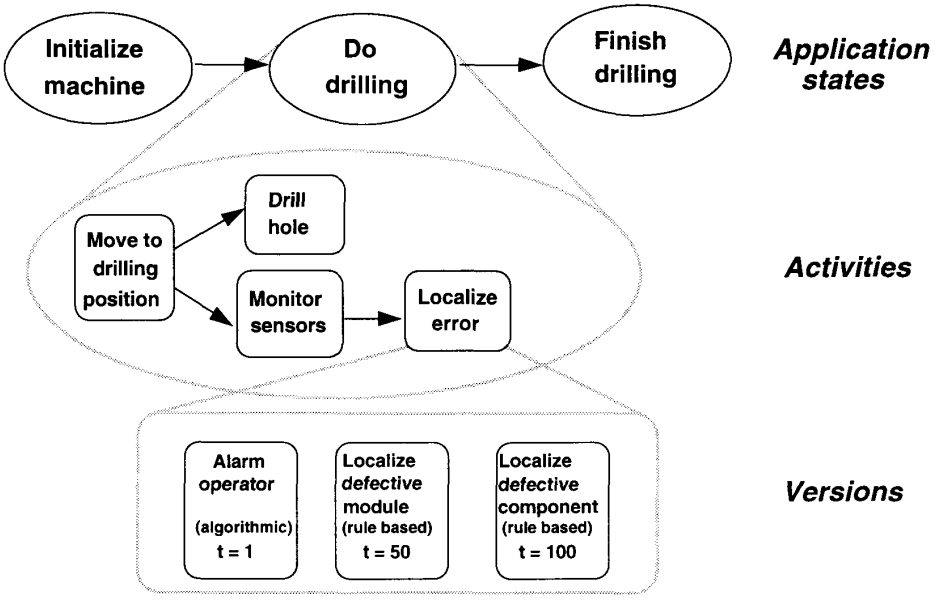

Figure 1. An IRTC application

For each application state there is a group of activities to be performed while in that state. An activity can be, for example, "drill a hole" or "update process display". For each activity there can be one or more activity versions, each of which implements the named activity, but taking different amounts of time and producing different qualities of result. For each activity version there is a corresponding task or function which is executed to implement the activity [because the activities can be modelled hierarchically, the lowest level activities are called terminal activities (TA) and corresponding versions terminal activity versions (TAV)]. The philosophy behind multiple versions and their management is explained later.

When an IRTC application is designed, the designer has to provide timing and other parameters for activities and activity versions as well as control information which is used for determining the system's behaviour depending upon the application state.

2.2. General architecture

The main components of the IRTC system are presented in Figure 2. The global controller, which controls the application at the highest level, consists of three main components: the Notifier, Scheduler and Context Processor. The Notifier is responsible for receiving and forwarding all the messages coming from application tasks. The Scheduler is responsible for managing schedules (i.e. ordered lists of runnable tasks): creating, modifying, and running them. It uses an Application Structure KB which contains information about activities and their versions (such as mutual dependencies and timing information). The Context Processor manages application states by receiving messages from application tasks for updating the application state vector and changing the application state when necessary. The knowledge about the conditions for application state changes is contained in Control KB.

The whole IRTC system - application and global controller - is run on the top of a commercial, POSIX 1004.3 compatible, real-time operating system (at the moment Solaris 2.3). For portability's sake, an interface, the Virtual Machine (VM), has been designed for accessing the system services of the real-time operating system.

2.3. Operation of the system

When the system enters a new application state, the Context Processor provides the Scheduler with a set of activities from which it creates a runable schedule. While creating the schedule, the Scheduler uses timing and other parameters from the activities and their versions. The schedule is adapted from a minimum schedule in which the shortest duration versions of the activities are used. While sufficient time is available,

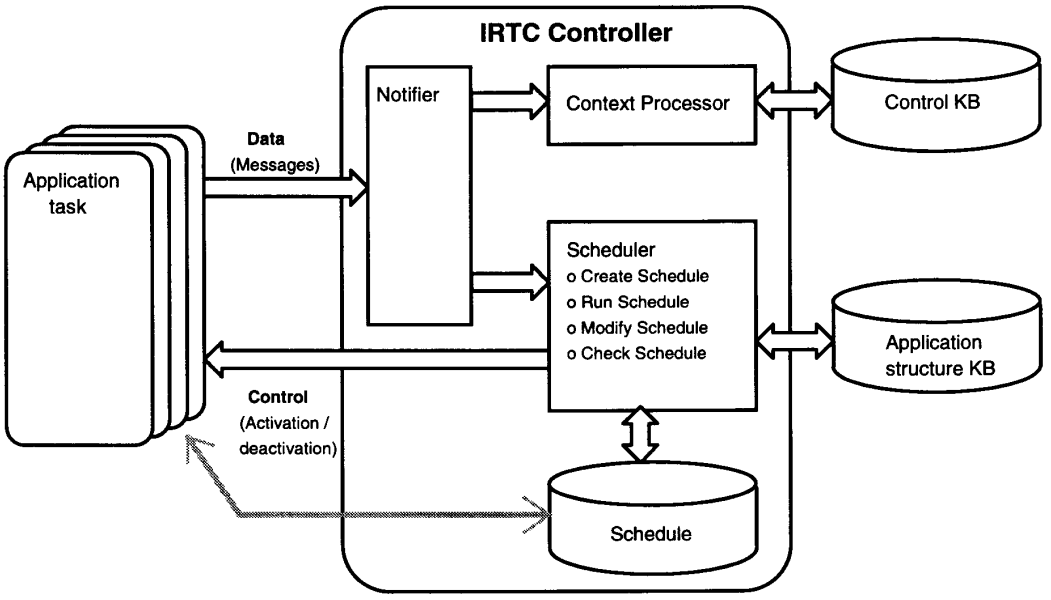

Figure 2. General architecture of the IRTC operating environment

longer duration versions are scheduled. The selection of which versions of which activities are used in the schedule depends on each activity's importance and each activity version's utility.

The Scheduler acquires a feasible schedule which it executes via its RunSchedule service. During execution, CheckSchedule monitors the timing of the execution. In case of a timing deviation, the CheckSchedule calls the ModifySchedule service which locally modifies the current Schedule. Modification could be the replacement of a longer duration version with a shorter duration version or vice versa or the addition or removal of an optional Terminal Activity. As the estimate for execution time of knowledge-based tasks is very rough, the available time is likely to change after the execution of each such activity.

The change of the application state is issued by Context Processor which receives messages from the application for updating the application state vector. Using the application state vector and control information provided by the designer, the Context Processor determines when to change the application state and to what state.

2.4. Activity versions & variable depth resolution

Variable depth resolution (VDR) involves the use of multiple methods for achieving a single purpose in an integrated system. Rapidly completing versions give acceptable answers in a short amount of time while longer duration, more complex versions give more complete, accurate, or useful results. VDR may be used to allow several activity versions to be able to handle the same objective, with one being selected depending upon both its features and available system resources. In our terminology each objective is handled by an activity, which contains one or more versions.

Variable depth resolution is a valuable technique for combining reasoning tasks with real-time systems to achieve real-time intelligent control of mechatronic systems. Multiple algorithmic tasks for the same job are also quite possible [4]. An algorithmic solution may be barely acceptable, but be guaranteed to run in a fixed amount of time, while a rule-based operation may do a lot more, but take an indeterminate amount of time that is significantly longer than the algorithmic method.

For example, an activity to handle a loss of pressure in a hydraulic fluid line might have an algorithmic version which alerts the operator and halts the drilling

Terminal Activity Handle_loss_of_hydraulic_pressure		Actions to take	Duration
Version 0	Algorithmic	Alert Operator and halt process	1
Version 1	Knowledge-Based	Localize break, reset values to close of faulty line and open parallel line, notify operator	50-100

Table 1. Two Versions of an Activity

(Table 1, Figure 3). A knowledge-based version might localize a break in the line, operate valves turning off the broken section, reroute fluid through a parallel redundant line, and alert the operator of the broken section which needs to be repaired -- all without interrupting the drilling process.

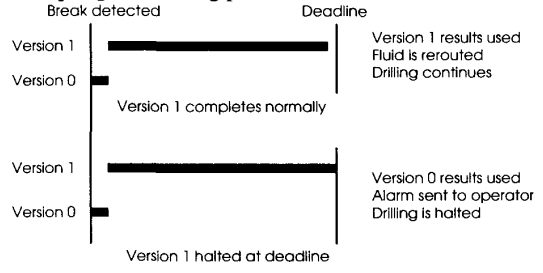

Figure 3. Acceptance of Results Based upon Duration of Knowledge-Based Version

In the simplest form VDR can be implemented so that there is a default version which takes a short known amount of time to complete, as well as a longer version which takes an unknown amount of time (the designer should provide an approximation of the duration). If there is enough time the longer version is tried first. If it has not finished before deadline minus the duration of the shorter version, then the longer version is stopped and the shorter version is executed in order to guarantee the achievement of a result from the activity.

Another possible way to use multiple versions is to execute successively more complex versions until either the available time has been consumed or there is no version available which could provide a better result. The result of each version is stored until it is replaced by a result from a more complex version or the activity is declared complete. This could be implemented by several distinct versions or by the same version being called each time (which means that it is iterative task) producing increasingly better results.

Variable depth resolution is related to variable depth reasoning, in which a single task has multiple methods for performing its objective and selects among them based upon its own judgement of the relevant timing issues. In real-time systems this would require the task to know about the existence of other tasks and their timing requirements as well as its own timing. We are using variable depth resolution instead, in order to keep all scheduling under the controller.

2.5 Scheduling

The scheduler views the system as being composed of coexisting activities, with an activity for each objective of the system. Each activity has a list of versions that could implement the activity, providing different qualities of responses and taking different amounts of time. Some versions could be rapid algorithmic versions while others are complex rule bases.

The controller maintains a list of activities that it may be instructed to schedule. With each activity is maintained the value of its importance and criticality as well as a list of the headers for the versions that are available for implementing that activity. Each version header contains information on the version's duration, its utility, its version type, and a means of accessing the version itself as well as arguments for the version and a couple of flags indicating how the version interacts with other versions for the same activity. The scheduler uses these parameters along with activity deadlines to select which versions to run. If there is but a single version in the list, no such processing occurs.

Criticality is tripartite: an activity may be mandatory, optional, or background. A mandatory activity must be executed before its deadline; this is not strictly true for an optional activity, although every attempt is made to do so; while background activities are run only when nothing else is schedulable. Importance is a number indicating the relative importance of completion of an optional activity. Utility is the relative benefit to the system of completion of different versions for the same activity. All of these values are assigned to the activities and versions by the system developer. The scheduler uses these parameters to dynamically determine the best way to schedule versions for the best operation of a continuous process.

If time permits, the scheduler may use progressive reasoning [5] for appropriately designed activities and schedule a more detailed version to replace the results of another version of the same activity that has already completed. Therefore, the controller maintains a list of conditional results and actions to take as the outcome of execution of its various activities. It puts entries into the lists when a result message is received from a version and implements the result or action by the deadline. If a result is received from a later version of the activity, the previous result is overridden.

3. Current situation

The system is currently under development. An application (a control system for a currently existing industrial machine) is currently being coded according to IRTC concepts. The first prototype of the IRTC control system became available for testing in February-March of 1994.

The most interesting question, for which the testing will give the answer, is the amount of the overhead caused by the global controller. It is the major potential drawback for the presented approach. Unfortunately, at the time of writing the paper, the system was not yet ready for measuring appropriately the timing characteristics.

4. Conclusion

Most of the systems dealing with real-time and knowledge engineering have concentrated on making a knowledge engineering system with real-time characteristics. In this paper a new approach was presented for integrating knowledge engineering activities within a real-time control system. Integration has been made possible by having multiple versions of one activity, and versions are selected dynamically according to the available time.

5. References

[1] Dean, T. & Broddy, M.; An Analysis of Time-Dependent Planning, Proceedings of the AAAI, 1988, pp. 49-54.

[2] Seppänen, P. & Foxvog, D.; IRTC - Project, Focusing on Real-Time Features of Expert Systems, Finnish Artificial Intelligence Conference, 9-11.6.1992, Otaniemi, Finland, pp. 90-95.

[3] Pfeffer-Thurat, N; IRTC: Une Plate-Forme pour le Developpement et l'Execution d'Applications de Controle Intelligent en Temps Reel, Genie Logiciel et Systemes Experts, September 1992, no. 28, pp. 70-73.

[4] Horvitz, E.; Reasoning Under Varying and Uncertain Resource Constraints, Prodeedings of the AAAI, 1988, pp. 111-116.

[5] Krijgsman, A. J. & Jager, R.; DICE: A Real-Time Toolbox, Artificial Intelligence in Real-Time Control (1992 IFAC/IFIP/IMACS Symposium), pp. 637-641.