A methodology for the development of distributed real-time control applications with focus on task allocation in heterogeneous systems

A. Prayati¹, C. Koulamas², S. Koubias¹, G. Papadopoulos²

¹ Dept. of Electrical Engineering & Computer Technology, University of Patras, Greece
² Industrial Systems Institute, 26500 Rio, Greece

prayati@ee.upatras.gr, koulamas@isi.gr, koubias@ee.upatras.gr, papadopoulos@isi.gr

Abstract* - A distributed application development methodology is necessary to define specific steps, through which the application specification can be successfully mapped to the system devices. Interoperability and real-timeliness are two major issues in distributed control application development. The main focus of this paper is a generic device model, developed to provide interoperability and the FBALL algorithm, defined to guarantee that real-time requirements are met. FBALL is a hybrid approach targeted for the nature of distributed control applications, resulting in an assignment of the application tasks to the system resources, as well as a feasible schedule that meets the real-time constraints. Based on these solutions to interoperability and real-time criticality, a methodology is presented, supporting the distributed control application specification, modeling and implementation to heterogeneous systems.

Key words - Distributed control, interoperability, real-time systems, scheduling, task allocation

I. INTRODUCTION

Today’s industrial control systems are characterized by special features like open field communication, distributed control architecture, a heterogeneous nature and last but not least, the hard real-time requirements that must be met in order to guarantee a successful application development. These features indicate specific needs in the development of control applications, like interoperability, and a correct partitioning and allocation of the application parts to the system resources. A lot of solutions have been proposed for the interoperability problem, whereas the need for an allocation technique has not yet been fulfilled.

* This work was supported by the European Committee Project TORERO (IST-2001-37573)
Interoperability is dealt in two levels, namely system interoperability and device interoperability. Device interoperability refers to the correct operation of a fieldbus, where devices from more than one vendor need to collaborate. The standard IEC61131 was the first attempt to provide a common terminology and reference model about HW and SW architectures, communication and languages for a special class of control device. Apart from the common device model, a general, common device description is also required for the representation of all device particular features of field devices. The inclusion of electronic device description language (EDDL) in the IEC standard IEC61804 [1] resulted to a common DDL perspective for device interoperability. Based on this common version of DDL, the IDA group made some extensions and created XDDML [2], combining DDL constructs with the XML syntax, allowing for the definition of new extensions without loss of compatibility. System interoperability addresses the need of distributed applications, relying on more than one fieldbuses, to communicate their different properties independently of the system specific particularities. Evolving standards such as the IEC 61499 [3] define the basic concepts for the design of modular, reusable, distributed components for industrial control systems. These standards define the function block (FB) construct as the main building block for industrial process measurement and control systems applications.

Besides interoperability, dealing with the real-time nature of control applications, is a very important issue. Real-timeliness is tightly coupled with automation control, from requiring strict synchronization times to imposing constraints to communication connections, rendering in this way, the communication part the most demanding issue in system implementation. The existing approaches [4], [5] in distributed control system development do not provide appropriate constructs to handle communication aspects of industrial systems, like network characteristics and device connection capabilities. Summarizing the solutions to the needs of distributed automation control systems, we conclude that an automated allocation solution is the missing part of the distributed application development. Moreover, in order to deal with the complexity of distributed applications, a general design trajectory, covering interoperability and real-timeliness, is necessary.

In this paper, a well-defined design methodology is presented, intended to support the distributed control application specification, modeling and implementation to heterogeneous systems. In section II, the methodology most important steps are defined, with main focus on the generic device model developed to support the system design and the FB allocation (FBALL) algorithm, which is extensively presented in
section III, for guaranteeing the real-time requirements. In section IV, a numerical example for a distributed control application development is presented, whereas in section V, the performance of FBALL is discussed and in section VI a summary of this research work is given.

II. DISTRIBUTED APPLICATION DESIGN METHODOLOGY

Application development includes all the different steps from requirements analysis and application specification to the implementation of its distributed parts to the system devices. The proposed methodology consists of the procedure depicted in Fig. 1. The innovations basically focus on the first five steps, whereas the two last consist of verification, thus debugging and testing, and device configuration.

![Fig. 1 Application development steps](image)

A. Application specification

The application requirements are extracted by the specific industrial process needs, translated into services and imposed constraints, analyzed in a formal specification format describing the application’s functionality. These requirements may be functional and communicational or performance constraints. Although on the sidelines of the IEC61804 standardization procedures, a definition of the intermediate steps towards a formal functional requirements description was attempted, the resulting representation, the so-called Functional Requirement Diagram (FRD), lacks the constructs to express nonfunctional requirements and constraints. In order to alleviate this weakness, the usage of the RT-UML (Real-Time Unified Modeling Language), as specified by the Real-Time Special Interest Group, is proposed. According to the RT-UML approach, every model is an abstraction of an aspect of the system, whereas the provided extensions of time semantics and
clocks, physical and logical resources and concurrency issues provide the necessary formal requirement specification notation. Thus, an RT-UML model is suitable for covering all functional and constraint information needed and can be used as a complete input to the next step of application modeling.

B. Application design

The application design model is based on the IEC61499 FB construct. According to IEC61499, a distributed application is defined by interconnected FBs, forming a FB Diagram (FBD). Designing the application refers to defining the relevant FBs of the FBD, specifically the FBs’ algorithms and execution control, their data and event connections and the time constraints. Based on the RT-UML diagrams, the FBD is constructed according to the guidelines defined following: For every activity diagram, a corresponding FB is selected. For every state of the state diagram and based on the class diagram, an algorithm is defined. IEC 61499 specifies neither implementation rules for the algorithm nor the programming language to be used. Consequently, the FB algorithm can be expressed in any programming language. For every transition of the state diagram, the event connections of the FBs are defined and based on the collaboration diagrams the corresponding data connections are made. The execution control algorithm is defined by the execution control chart (ECC), which determines one or more actions associated with a state. For every time construct, the appropriate constraints are inserted to the connections or function execution times. Newly introduced XML elements represent these time bounds.

C. Requirements for system characteristics

At this phase of the application development, the system characteristics are described, in order to provide information that influence the implementation of the application design. If the system infrastructure is predefined, the system characteristics are described based on the given devices. On the other hand, if there are no restrictions on the underlying system resources, a library with all available devices provides the required resources for the application implementation. In both cases, the system is modeled in terms of the properties of the network and the system devices. For the device model description, a generic UML model has been developed [6], as shown in Fig. 2. The device model presented by IEC61131 is destined for PLCs and does not address all the concepts required for distributed application development, like communication, network interface, constructs that are essential for meeting real-time constraints.
Fig. 2 Device model

In order to meet the requirements of interoperability between heterogeneous real-time systems, as well as the need for a unified and seamless application development methodology, the device model in [6] is based on the IEC61499 constructs and covers all device aspects, as identification, communication, functionality definition and configuration, in a “gray box” model, where unnecessary details are encapsulated.

The system is described by a diagram of interconnected devices, where the device representatives correspond to a device model instance, as shown in Fig. 3, where there is a direct relation between the device model and the textual device description from the XDDML file.

Fig. 3 System Diagram

The information for the device model is also specified in a XDDML textual representation, which define

Device Identity containing information for the device unique identification, the Application Process
describing the device operations, attributes and services with regards to the installed application, the *Device Functions* and the *Device Manager* used for configuration and monitoring when the application is integrated in the device.

**D. Allocation of distributed application to devices**

Meeting the functional, communication and real-time requirements of the application specification is the goal of this phase, where the distributed application tasks must be allocated to the system devices. Based on the application requirements and on the system properties, the allocation step constitutes the assignment of the FBs of the FBD to the devices resources. The output of the allocation step results in an allocation and scheduling scheme, namely a Gantt chart, where the release and finish times of the FBs are defined for the resources, to which they are allocated. The allocation provides the optimal distribution of control, taking into account the storage, computational and real-time restrictions imposed to the functional and communication application requirements. More precisely, we discriminate between the different compatibility factors.

1) *Industrial Process Connections*: The industrial process interface characteristics of the device must be compatible with the application parameters, I/O channel connections and data types. For every FB assigned to a device, its variable types are checked for compatibility with the device’s I/O.

2) *Storage Capacity*: As far as memory is concerned, the storage capacity of the resource must meet the application’s memory needs. For this purpose, the number of FB variables and their types must be used to calculate the required memory size, which must be less than the resource’s RAM size. More precisely, the resource’s storage capacity is consumed by the OS tasks, the device functions, as well as the FBs that will be allocated to it. In that sense, the memory requirement of the FB algorithm must be less than the resource’s storage capacity, after extracting the capacity dedicated to the OS and to the device functions.

3) *Computational Efficiency*: The complexity of the FB algorithms and the computational power of the target-resource are crosschecked at this point. For the computation of a FB in a resource, the computational capacity of the resource should be large enough to support the computational needs of the FB algorithm. The algorithm complexity is expressed by the amount of data that needs to be processed; that is the amount of data initially required in the resource plus the bits supplied during its execution. Considering a RAM memory type, the computational capacity of the resource, reserved for the specific FB algorithm execution, depends on the allocated size $S$ in the resource foreseen initially for the start of the algorithm
execution, as well as on the run-time computational needs. In total, the computational capacity of the resource is calculated as \((S + T \times b)\), where \(b\) is the number of bits of the algorithm input variables to the CPU and \(T\) is the number of cycles needed for their processing.

4) **Real-time constraints**: An optimal assignment of the FBs to the system devices is required for meeting the application real-time constraints. FBALL was developed, suitable for the particular features that automation control systems impose, under the assumption that the FBs composing the application are tasks and the FBD may be mapped to a task graph. The Branch and Bound (B&B) [7] and the clustering [8] techniques are the most known ones for solving the task allocation problem based on task graphs. The B&B algorithm has a disadvantage of a large search space, where all combinations of allocation are considered. The clustering technique has the drawback that scheduling is performed based on estimated approximation of the execution and communication times of the tasks, making it inexact for real-time performance analysis. FBALL is proposed as a hybrid allocation technique, where the vertexes of the B&B tree represent the assignment of clusters of tasks instead of one single task to the system resources. FBALL eliminates the two aforementioned drawbacks, by reducing the search space of the B&B tree and shifting the scheduling decision when the allocation is known and the exact values of the execution and communication times may be used.

### E. Device programming

At this stage of application development, the devices are programmed in order to implement the allocated application parts. The implementation consists of source code generation for device communication and algorithmic behavior. In [12], we define an aspect-oriented programming (AOP) approach for weaving the communication related code into the distributed control application code by means of AspectJ, an extension for aspect-oriented programming with Java. Code generation for functional implementation refers to defining the algorithmic part of the device, meaning either a parameter configuration for the instantiation of supported FB types or a compilation of the part of the allocated FB diagram, in case the FB type does not exist in the device.

**III. THE FBALL ALGORITHM**

An overview of the FBALL algorithm is provided in Fig. 4, together with the defined notation to facilitate its description and analysis.
First, the tasks are assigned priorities, taking into account the end-to-end deadline imposed to the task graph, the task precedence constraints and an estimation of the execution and communication times of the task graph. Next, tasks are clustered under the objective to minimize the communication load and a B&B algorithm is applied for the resulting clusters. For the B&B procedure, we introduce a preliminary step, where the search tree is pruned according to compatibility checks between the tasks’ requirements and the resources’ capabilities. Finally, an optimal assignment scheme is pointed out with reference to the scheduling procedure, whose criterion is the minimum system finish time.

### A. Priority Assignment

The ASSIGN_PRIORITIES procedure is shown in Fig. 5. For the priority assignment the slicing technique is applied for the end-to-end deadline distribution with respect to the critical path of the task graph.

**Find critical path:** The critical path is identified based on metric $R$, which expresses the laxity for the different paths in the graph regarding the end-to-end deadlines. In order to apply $R$ to our real-time requirement problem, communication weights are considered in the task allocation analysis, forming $R$ as

$$R = (D - \sum \bar{\epsilon}_i - \sum \text{com}_{ij})/n$$

(1)

The relative deadline for each task $\tau_i$ in the path is

$$d_i = \bar{\epsilon}_i + R$$

(2)

The execution time threshold and the virtual execution time are the two concepts introduced by [10], in order to improve the performance of the slicing technique. Accordingly to the execution time analysis, the concepts of communication time threshold and virtual communication time are defined.
The virtual communication time aims to assign extra laxity to tasks with high communication load, whether this is caused by heavy data load or by low network connection quality.

\[
\text{com}_{ij} = \begin{cases} 
  c_{i,j} & , c_{i,j} < C_{\text{th}} \\
  c_{i,j} (1 + k_c \xi_c / b) & , c_{i,j} \geq C_{\text{th}}
\end{cases}
\]  

(3)

where the surplus factor \(k_c \xi_c / b\) expresses the amount of laxity by which the communication time may be increased according to the locally adaptive laxity ratio. \(\xi_c\) stands for the set of parallel communication edges, defined as the number of communication edges that are not in the same path as \(c_{i,j}\). After calculating the metric \(R\), the critical path is determined by the path that minimizes \(R\). Distribute end-to-end deadline: The end-to-end deadline is distributed as described above to the tasks in the critical path. The relative release time of a task \(\tau_i\) in the critical path is

\[
r_i = d_j + \text{com}_{ij}
\]  

(4)

The absolute release time \(r_{ai}\) of task \(\tau_i\) is defined as the sum of the relative release times of its preceding tasks in the path, including task \(\tau_i\) itself, expressed as

\[
r_{ai} = \sum_{j=1}^{i} r_j
\]  

(5)

The absolute deadline of task \(\tau_i\) in the critical path is the sum of the relative deadlines of the tasks preceding \(\tau_i\) in the path and the communication time between task \(\tau_i\) and its immediate predecessor \(\tau_{i-1}\). For the absolute deadline of task \(\tau_i\), it holds

\[
d_{ai} = \sum_{j=1}^{i-1} (d_j + \text{com}_{j,j+1}) , i \geq 2
\]  

(6)
For the start task $\tau_i$ of the path, the absolute deadline is defined as the sum of the relative deadline of the task and its absolute release time, thus

$$d_{ai} = d_i + r_{ai} \quad (7)$$

**Remove critical path:** After the end-to-end deadline has been distributed to the tasks in the critical path, these latter tasks are removed from the graph and the procedure is repeated again until no tasks are left in the graph. At every iteration, the metric $R$ must be recalculated for the remaining paths, requiring the deadline $D$ of equation (1) to be formed as the difference of the absolute deadline $d_{a(\text{end})}$ of the end task in the path and the absolute release time $r_{a(\text{start})}$ of the start task of the path, holding

$$D = d_{a(\text{end})} - r_{a(\text{start})} \quad (8)$$

For the remaining paths, these parameters, namely the absolute deadline of the end tasks and the absolute release time of the start tasks, must be computed before proceeding with the whole procedure again. The absolute release for the task $\tau_i$ that does not belong to the critical is calculated as the sum of the absolute deadline of its predecessor $\tau_j$ and the communication time between $\tau_i$ and $\tau_j$. If the task has more than one predecessors, the maximum sum of the predecessors is selected. Thus, the absolute release time of task $\tau_i$ not belonging to the critical path is

$$r_{ai} = \max \{(d_j + \text{com}_{j,i}) : \tau_j < \tau_i \} \quad (9)$$

The absolute deadline of task $\tau_i$, not belonging to the critical path, is set to the earliest release time of its immediate successor $\tau_j$ in the critical path, minus the communication time between the tasks $\text{com}_{i,j}$

$$d_{ai} = \min \{(r_{aj} - \text{com}_{i,j}) : \tau_j < \tau_i \} \quad (10)$$

Once all tasks have been assigned deadlines, they are given a priority level equal to the absolute release time.

**B. Clustering**

The tasks are sorted in decreasing order of their absolute release times. Clustering is applied at this phase to the sorted tasks according to priority levels, as shown in the pseudocode of Fig. 6. The main objective of this step is to cluster tasks with high communication cost but not the cost of extra computational load. In this context, Ramamritham [11] defines a communication factor as

$$\rho_{ij} = (\bar{e}_i + \bar{e}_j) / \text{com}_{ij} \quad (11)$$
\( \rho \) has a range between zero (0) and the maximum value of \( \rho \) corresponding to the less tight communicated tasks of the graph.

```plaintext
CLUSTER_TASKS (task graph, release times, sorted_task_list) {
  CSsize_thres <- LCM of all task periods;  //cluster size threshold
  cf_thres = Calculate communication factor threshold(task_graph);
  task \( t_i \) <- Get first task from sorted_task_list;
  \( \text{while} \) (checked\_tasks > 0) {
    in_set <- Find fan-in tasks(\( t_i \));
    if (task not clustered) {
      Form new cluster \( C_k \) = \{\( t_i \} ;
      cluster\_size += \( e_i \);  
      checked\_tasks--;
    }
    for every Task \( t_j \) from in_set {
      com\_factor = \( (e_i + e_j) \)/Com\_wt;
      if (cluster\_size + \( e_i > \text{CSsize_thres}\) \&\& (com\_factor > cf\_thres)) {
        Add \( t_j \) to \( C_k \);
        cluster\_size += \( e_j \);
        Mark task as clustered;
        Zero communication time between tasks;
        \( \text{Decrease checked tasks} \);
      }
    }
    ASSIGN PRIORITIES (task graph, end-to-end deadline);
    Sorted\_Task\_list <- Sort tasks in increasing order of their release time;
    return cluster\_list;
  }
}
```

**Fig. 6** The procedure for clustering of tasks

Clustering will decide upon a threshold value of \( \rho \), whether the tasks will be assigned to the same cluster or different ones. The clustering decision is also based on the size of the cluster, which must not exceed the cluster size threshold \( S_{th} \). \( S_{th} \) is defined to be the hyperperiod, as the least common multiple of the periods of all tasks in the graph. The task periods are expressed as an integer, assuming them a multiple of the system clock cycle. If the size of the cluster does not exceed the cluster size threshold \( S_{th} \), the two tasks are clustered together.

**C. Branch & Bound on the clusters**

Since the clusters formed in the previous step are considered for allocation to one resource each, they may be considered as supertasks at this stage. The B&B is applied as described in Fig. 7.

```plaintext
BRANCH TEST (clusters, pruned tree) {
  //Calculate best-case cluster release and finish time
  for every resource req. {
    if (req supports all clusters) {
      selected\_req <- resource with the minimum execution time;
    } else {
      selected\_req <- resource with the minimum communication time;
    }
  }
  for every cluster \( C_k \) {
    \( P_{ran} \) <- Calculate worst-case release time (\( \rho \));
    \( P_{ran} \) <- Calculate worst-case finish time (\( \rho \));
    \( S_{ran} \) <- Calculate worst-case finish time (\( \rho \));
    for every branch \( b_i \) of the pruned tree ( \( C_k \)) {
      is\_assigned <- \( C_i \) in \( C_k \);
      \( S = \sum e_i / \text{Cluster size} \);
      \( \text{Cluster absolute release time with } t_i \text{, start task of } C \);
      \( P_{ran} \) <- \( C_i \) in \( C_k \);
      \( \text{Cluster finish time } F_i \text{ with } t_i \text{, end task of } C \);
      \( P_{ran} \) <- \( \text{Finish time of } C_i \text{ allocated last in } \text{is\_assigned} \);
      Allocation\_sequence <- \( b_i \) accepted);
    }
    return Allocation\_sequence;
  }
}
```

**Fig. 7** The Branch and Bound test procedures
**Form the search tree:** The tree is formed by vertexes representing at each level the assignment of a cluster to the different system resources. Each vertex is expanded by considering all possible assignment of the subsequent clusters, forming leaves that represent the complete allocation alternatives.

**Prune the search tree:** In order to simplify the search space even further, before proceeding to the next stage, we exclude the vertexes with incompatible characteristics between the clusters and the system resources. More specifically, based on the afore mentioned compatibility factors, we perform checks with reference to:

- the exclusive vector formed based on the XML specification for the FBs, which among others describe the resource types that can support the implementation of the specific FBs.
- the industrial process parameters forming the I/O for the clusters and the resource physical connections.
- the storage capacity of the resource and the application’s required memory size.
- the complexity of the task algorithms and the computational power of the target-resource.

Based on these three checks, the tree vertexes are pruned and the tree is ready for the branching test.

During the BRANCH_TEST procedure, the optimal path is quested based on the criteria of execution time and latest completion time. The BRANCH_TEST is performed by calculating the best and worst case release and finish times of the clusters under the assumptions of [9]. For the best- and worst-case analysis, the assignment is known and the execution time represents the actual value of the processing time of the task.

The absolute release time of task $\tau_i$ is the sum of the execution time of all its predecessors in cluster $\text{Cl}_i$

$$ r_{ai} = \sum_{j=1}^{i-1} e_j, \quad \tau_j \in \text{Cl}_i \quad (12) $$

For the release time of the start task of the cluster, we assume that it is zero (0), if the task is the start task of the task graph. If the start task of the cluster has precedence constraints with other tasks, its release time is the maximum of the absolute finish time of the predecessors plus the communication time between them.

$$ r_{a(\text{start})} = \begin{cases} 0, & \tau_i \text{ has no precedence constraints} \\ \max \left\{ \left( r_{aj} + c_j + c_{ij} \right) : \tau_j < \tau_i \right\}, & \tau_i \text{ has predecessors} \end{cases} \quad (13) $$

Based on these calculation, we calculate the size $S_i$, the release time $R_i$ and the finish time $F_i$ for every generated cluster $\text{Cl}_i$ respectively. The release time of the cluster equals the release time of the start task of the cluster. The finish time of a cluster is defined as the finish time of the end task $\tau_j$ in the cluster, expressed as the sum of its absolute release time and its execution time.
The latter equations hold under the assumption we performed clustering, that the tasks in a cluster are executed in a sequential order and that communication time between together clustered tasks is zero. For every vertex, the size $S_i$, release time $R_i$ and finish time $F_i$ are calculated for every generated cluster $C_l$, and based on these values, the branching test is performed. The set of equations (12)-(14) hold with the difference that in this case, the execution and communication time values of the predecessor are the virtual values, since at this point the allocation of the precedent cluster is not known.

The BOUND_TEST procedure decides on the optimal allocation and scheduling scheme based on the calculation of the objective function defined as the probability to meet the corresponding to the vertex task deadline. The calculation of this probability is based on the calculation of the cluster lateness, which is the difference between the cluster completion time and its absolute deadline. The vertex with the highest probability is selected, pointing the optimal allocation scheme. If the optimal probability corresponds to more than one allocation alternatives, the one with the earliest system finish time is selected.

### D. Scheduling

As shown in Fig. 8, the SCHEDULE procedure is applied after sorting clusters in increasing order of their release times. The cluster with the highest release time is picked and its tasks are assigned to the corresponding resource. The tasks of the cluster are sorted in increasing order of release time and the task with the smallest release time is selected to be added to the scheduling scheme of the resource. If other tasks are already allocated to the resource with the same release time, then the task with the earliest finish time is scheduled first.

The schedule’s feasibility is defined as the probability that the tasks are completed before their deadlines. The schedulability of task $\tau_i$ is expressed by the ratio of the laxity to the absolute deadline $d_{ai}$, as

$$P_{\tau_i} = \frac{u(d_{ai} - f_i)}{d_{ai}}$$  \hspace{1cm} (15)

The probability expressing the feasibility of the whole task schedule is defined as the multiple of the probabilities expressing the task’s schedulability, thus

$$P = \prod_{\tau} P_{\tau_i} = \prod_{\tau} \left\{\frac{u(d_{ai} - f_i)}{d_{ai}}\right\}$$  \hspace{1cm} (16)
Fig. 8 The SCHEDULE procedure

The CALCULATE_RESULT procedure specifies the release and finish times, based on the execution and communication time values for the given allocation scheme, on the schedule in every resource, while taking into account the precedence constraints in the other resources. The absolute deadline of a task \( \tau_i \) is the minimum of the absolute deadline of its successors minus the predecessors’ execution time and the communication time between them, for the successors assigned to a different resource than task \( \tau_i \). This result is also compared to the absolute deadline of the successor of task \( \tau_i \) scheduled in the same resource, minus the successor’s execution time. The final value of task’s \( \tau_i \) absolute deadline is the minimum of these two latter values, thus

\[
d_{a_i} = \min \left\{ (d_j - e_j), (d_l - e_l - c_{l,i}) : \tau_j \text{ scheduled last before } \tau_i, \forall \tau_l < \tau_i \right\}
\]  

(17)

The CALCULATE_RESULT procedure uses the \textit{suspend} property defined for every task to express the validity of the release time value considering the precedence constraints. If the \textit{suspend} property is set to zero (0), then the task has no predecessors pending with respect to their release time and thus its final release time may be computed.

IV. NUMERICAL EXAMPLE

The distributed control application design methodology is applied to a system development problem. The industrial process, whose P&ID according to IEC61804 is shown in Fig. 9, represents the causticizing procedure of a typical Kraft pulp mill. Green liquor from the recovery area is converted to white liquor through the addition of quicklime in the shaker. For maximum conversion without overuse of lime, the ratio
of lime added per unit of green liquor and shaker bowl temperature must be maintained. The ratio of lime to
green liquor may be automatically adjusted based on analyzers after the shaker to compensate for variations
in the liquor and lime quality. In case the green liquor flow to the shaker is stopped, the lime feeder motor
should automatically be turned off as a safety precaution.

Following the proposed methodology, the FBD diagram is extracted based on the application specification.
For simplicity reasons, the FBD is presented in a high abstraction level, with some instances of the FBs for
Analog Input (AI), Analog Output (AO) and PID in Fig. 9. The corresponding task graph and the system
diagram are shown in Fig. 10. The table in this figure also shows the required information for the two
diagrams as far as the application of the FBALL algorithm is concerned. This information is extracted from
the XML and XDDML description of the FB and the system diagrams respectively. The exclusive vector for
every task is formed with a row of a number of elements corresponding to the number of resources in the
given system platform, the element at each column set to 1 if the corresponding resource does not support
the specific task. The execution time for every task is given as an array, whose elements at every column
correspond to the existing resources. The network delay for every task is given as an array, whose indexes
correspond to the number of tasks in the task graph and the values refer to the network communication cost
of the current task to its successors.
The presented FBALL algorithm was implemented in Java and applied to a number of different task graph examples. Four cases of FBALL were tried out for examples with different numbers of tasks, considering as
objective the success ratio of the algorithm search space; that is the ratio of number of assignments, schedulable within the time constraints, to the total number of possible assignments of the search tree. The four cases correspond to the combination of use and no use of the PST and clustering respectively, as shown in the left schema of Fig. 12. The use of priority slicing technique and clustering before the application of B&B results in an increase of the success ratio by a factor of 10% compared to the simple use of B&B without application of the introduced steps of the FBALL.

![Fig. 12 Success ratio and feasibility results for the FBALL algorithm](image)

The use of PST has little impact on the increase of the success ratio, it does, however, improve the feasibility of the resulting schedules by a factor of 15% as shown in the right schema of Fig. 12. This is due to the fact that PST handles more efficiently the deadline distribution to the task graph, taking into account the laxity and thus, increasing concurrency and allowing a better performance of scheduling.

VI. CONCLUSION

In this paper, a distributed control application development methodology was presented, defined especially for heterogeneous systems where real-time constraints are a major issue. The proposed methodology defines all intermediary steps, through which the control application specification can be successfully distributed to the system devices. In order to meet the distributed control application development requirements, a generic device model was developed, which in combination with the IEC61499 FB construct provides interoperability. Moreover, the FBALL, a static, off-line, nonpreemptive allocation and scheduling algorithm is presented, for guaranteeing the real-time performance of the final system implementation. The proposed
approach forms clusters of tasks and applies a Branch and Bound search in order to find the optimal allocation and scheduling scheme for the FBs to the system resources. This consideration eliminates the drawback of the clustering technique, by postponing the calculation of relevant parameters after scheduling, when no estimation is required and the real values are used. Moreover, it reduces the search space of the B&B algorithm, since the number of clusters is smaller than the number of tasks in the task graph.

VII. REFERENCES