Volume 7, Issue 3, March 2017 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Example-founded Package Conception of Real-Time Embedded Arrangements

[1]Ayasha Siddiqua, [2]Shaista Sabeer, [3]Nivedita Soni
[1]Lecturer Jazan University, Saudi Arabia
[2] Lecturer Jazan University, Saudi Arabia
[3]Technical Analyst HCL

DOI: 10.23956/ijarcsse/V7I3/01321

***Abstract:** This research paper depicts a example-founded package aim method acting acting for real-time embedded arrangements. When planning coincidental and real-time arrangements, it is necessity to blending object-oriented conceptions with the conceptions of coincident treating. This research paper depicts a example-based package aim method acting acting for planning real-time embedded arrangements, which incorporates object-oriented and coincidental processing conceptions and uses the UML annotation.*

***Keywords:** real-time, UML, concurrency, embedded package, aim method acting, package package lines.*

I. INTRODUCTION

With the heavy change in the price of microcomputer and semiconductor device chips and the big gain in microcomputer functioning over the last few long time, real-time and distributed real-time microcomputer founded arrangements are a very cost-efficient result to many troubles. Nowadays, more and more commercial-grade, developed armed forces, medical, and user products are personal computer based and either package assured or have a crucial package constituent to them. This research paper depicts a example-founded package aim method acting for real-time embedded arrangements.

Real-time arrangements are reactive arrangements, so that control decisions are often state dependent, hence the importance of finite state machines in the aim of these arrangements. Real-time arrangements typically need to process concurrent inputs from many sources, hence the importance of concurrent package aim. They have real-time throughput and/or response time requirements, so there is a need to analyze the Functioning of real-time aims. Furthermore, there is a need to integrate real-time technology with modern package engineering concepts and method actings.

In example-founded package aim and growth, package patterning is employed as an necessity part of the package growth cognitive process. Examples are constructed and examined anterior to the accomplishing of the arrangement, and are wont to conduct the accompanying effectuation. A better agreement of an arrangement can be got by conceiving the multiple views [17, 18], such as necessities examples, static examples, and active examples of the arrangement. A graphical exampleing language such as the Unified Exampleing Language (UML) helps in developing, understanding and communicating the different views.

This research paper furnishes a summary of planning real-time engrafted packages. It begins by furnishing a summary of coincidental auctioning conceptions in Section 2. In section 3, semantic error support for coincidental and real-time arrangements is in short talked about. Section 4 demonstrates a summary of coincidental and period aim method acting acting. With this background, a summary of an example-founded package aim method acting acting for administered and period engrafted arrangements inclines in Section 5. The COMET method acting acting [15] incorporates object-oriented and coincidental auctioning concepts, and employs the UML notation. This research paper depicts all aims with the UML 2.0 notation [5,31]. Section 6 depicts package architectural conventions for real-time assure. Section 7 depicts the execution psychoanalysis of real-time package aims. Section 8 depicts the aim of real-time engrafted package package lines.

II. COINCIDENT AUCTIONING CONCEPTIONS

A property of all real-time embedded arrangements is that of coincident auctioning; that is, many actions coming about at the same time whereby, often, the order of entering consequences is not foreseeable. Accordingly, as real-time embedded arrangements administer with several coincident actions, it is highly suitable for a real-time embedded arrangement to be integrated into concurrent jobs (also known as concurrent processes or threads). This section depicts the conceptions of the concurrent project, and the communicating and coordination among co-operating projects.

A coincident job constitutes the carrying out of a consecutive curriculum or serial constituent of a coincident curriculum. A coincident arrangement comprises of several jobs accomplishing in parallel. Each job administers with one sequential blade of carrying into action. Concurrency in a package is found by having multiple asynchronous jobs, campaigning at different speeds. From time to time, the jobs need to convey and synchronize their functioning's with

each other. The concurrent jobbing conception has been employed extensively in the conception of operating arrangements, real-time arrangements, interactive arrangements, distributed arrangements, parallel arrangements, and in computer simulation applications [2].

Conceive the case of a robot arrangement that assures up to four robot arms. Generally, there is one robot program to assure the functioning's of one robot arm. Thus, each accomplishing robot curriculum is considered a package job.

In most coincident arrangements, the jobs need to co-ordinate their actions. Two directions in which this procedure can be accomplished are mutual exception and job synchronizing [2].

III. RUN-TIME ACCOMPANIMENT FOR COINCIDENT JOBS

Runtime accompaniment for coincidental auctioning may be provided by

- Kernel of an OS. This has the practicality to allow for avails for coincident auctioning. In some modern OS, a microkernel allows for minimal utility to support coincidental processing, with most services provided by arrangement level jobs.
- Runtime network for a coincident language.
- Threads package. Provides services for dealing threads (lightweight processes) within heavyweight actions.

IV. APPRAISE OF AIM METHOD ACTING ACTING FOR COINCIDENT AND REAL- TIME ARRANGEMENTS

For the aim of coincident and real-time arrangements, a major part came in the late 70s with the foundation of the MASCOT annotation [34] and later the MASCOT aim solution [35]. Established on a data flow access, MASCOT validated the way jobs intercommunicate with each other, via either canalizes for message communicating or pools.

The 1980s saw a cosmopolitan growth of package aim method acting acting, and several arrangement aim method acting acting were brought in. Parnas's act upon with the Naval Research Lab in which he researched the employ of data hiding in large-scale package aim led to the growth of the Naval Research Lab (NRL) Package Cost Reduction Method acting. Work on employing Structured Analysis and Structured Aim to coincident and real-time arrangements led to the growth of Real-Time Structured Analysis and Aim (RTSAD) [22,41] and the Aim Approach for Real-Time Arrangements (DARTS) [13] method actings.

Another package growth method acting to issue in the former 1980s was Jackson Arrangement Growth (JSG) [23]. JSG was one of the first method acting actings to advocator that the aim should example realism first and, in this respect, preceded the object-oriented analytic thinking method acting actings. The arrangement is conceived a computer simulation of the real world and is projected as an electronic network of coincident jobs, where each real world entity is molded by entails of a coincident job. JSG also held the then established concocting top-down aim by encouraging a middle-out behavioral approach to package aim. This access was a precursor of object interaction example, an essential expression of modern object-oriented growth.

The former object-oriented analytic thinking and aim method acting acting's underlined the geomorphologic expressions of package growth through data hiding and inheritance but failed the active aspects. A major part by the Object Exampleing Proficiency [30] was to distinctly evidence that active exhibiting was equally significant. In addition to bringing in the static exampleing annotation for the object plots, OMT demonstrated how active exampleing could be executed with state graphs (hierarchical state transition plots originally conceived by Harel [21]) for demonstrating the state-dependant behavior of active aims and with sequence plots to show the sequence of interactions among objects.

The CODARTS (Concurrent Aim Approach for Real-Time Arrangements) method acting acting [14] rested on the intensities of in the beginning coincident aim, real-time aim, and early object-oriented aim actings. These admitted Parnas's NRL Method acting, Booch Object-Oriented Aim [4], JSD, and the DARTS acting by accenting both data hiding module structuring and job structuring. In CODARTS, concurrency and timing consequences are considered during job aim and data hiding consequences are conceived during module aim.

Octopus [1] is a real-time aim method acting acting established on use cases, static molding, aim interactions, and tategraphs. By blending conceptions from Jacobson's employ cases with Rumbaugh's static patterning and tategraphs, Octopus expected the blending of the annotations that is now the UML. For real-time aim, Octopus aims detail emphasis on interfacing to external devices and on job structuring.

ROOM – Real-Time Object-Oriented Patterning – [38] is a real-time aim method acting acting that is closely tied in with a CASE (Computer Assisted Package Engineering) tool called Object Time. ROOM is founded around actors, which are active objects that are exampleed employing a variation on state graphs called ROOM graphs. A ROOM example, which has been defined in sufficient detail, may be accomplished. Thus, a ROOM example is functional and may be used as an early example of the arrangement.

Buhr [6] brought in a concerning concept called the use case function (based on the use case conception) to address the consequence of active example of large-scale arrangements. Use case maps consider the sequence of interactions among objects (or aggregate objects in the form of sub arrangements) at a larger grained level of detail than do communicating plots.

For UML-established real-time package growth, Douglass [10, 11] has allowed for a comprehensive examination verbal description of how UML can be applied to real-time arrangements. The 2004 book depicts applying the UML notation to the growth of real-time arrangements. The 1999 book is a detailed compendium covering a wide range of topics in real-time arrangement growth, including safety-critical arrangements, interaction with real-time operating arrangements, real-time scheduling, behavioral patterns, and real-time frameworks, debugging, and testing.

V. EXAMPLE-BASED PACKAGE AIM METHOD ACTING FOR CONCURRENT AND REAL-TIME EMBEDDED ARRANGEMENTS

5.1 Introduction

Most texts on object-oriented analysis and aim only deal the aim of sequential arrangements or exclude the significant aim consequences that need to be covered when aiming real-time and distributed application program [2, 14]. It is necessitate blending object-oriented conceptions with the conceptions of concurrent auctioning in order to successfully aim this application program. This research paper depicts some of the key aspects of the COMET example-based package aim method acting for real-time embedded and distributed arrangements. COMET integrates object-oriented and concurrent processing conceptions and uses the Unified Modeling Language (UML) notation. It also depicts the decisions made on how to employ the UML annotation to address the aim of coincident, allotted and real-time embedded placements. Examples are given from a Pump Control System, which is described using the UML 2.0 notation.

5.2 The COMET method acting

COMET is a Coincident Object Exampling and Architectural Aim Method acting for the growth of concurrent applications, in particular allotted and real-time embedded coatings [15]. As the UML is now the exchangeable annotation for depicting object-oriented examples [5,24,31], the COMET method acting employs the UML notation throughout. The COMET Object-Oriented Package Life Cycle is extremely iterative. In the Necessities Exampling phase, a use case example is developed in which the operational requirements of the placement are defined in terms of actors and use cases.

In the Analysis Exampleing phase, static and active examples of the system are arisen. The static example determines the geomorphologic relationships among problem domain categories. Object structuring standards are wont to decide the objects to be considered for the analysis example. A active example is then arose in which the use cases from the necessities example are realized to show the objects that take part in each use case and how they interact with each other. In the active example, state dependent objects are defined using state graphs.

In the Aim Exampling phase, an Architectural Aim Example is arisen. Sub arrangement structuring criteria are furnished to aim the overall package architecture. For distributed applications, a component based growth approach is taken; in which each sub arrangement is aimed as a distributed self-contained constituent. The accent is on the division of province between clients and servers, admitting consequences concerning the consolidation vs. distribution of data and control, and the aim of message communicating ports, admitting synchronous, asynchronous, brokered, and group communicating. Each coincident sub arrangement is then planned, in terms of active objects (jobs) and passive aims. Job communicating and synchronization ports are determined. The Functioning of real-time aims is estimated using an access based on rate monotonic analysis [37].

Discerning characteristics of the COMET method acting are the vehemence on:

- Coordinating standards to assist the architect at dissimilar stages of the analysis and aim process: subarrangements, objects, and concurrent jobs.
- Active exampling, both object communicating plots and state graphs, describing in detail how object communicating plots and state graphs relate to each other.
- Distributed application aim, addressing the aim of configurable distributed components and inter-component message communicating ports.
- Concurrent aim, addressing in detail job structuring and the aim of job ports.
- Functioning analysis of real-time aims using real-time scheduling COMET emphasizes the use of structuring criteria at different stages in the analysis and aim process. Object structuring criteria are used to help determine the objects in the arrangement, sub arrangement structuring criteria are used to help determine the subarrangements, and concurrent job structuring are used to help determine the jobs (active objects) in the arrangement. UML stereotypes are used throughout to clearly show the use of the structuring criteria.

The UML Annotation affirms Requirements, Analysis, and Aim conceptions. The COMET method acting breaks requirements, analysis, and aim actions. Requirements exampling address determining the operational necessities of the arrangement. COMET differentiates analysis from aim as follows. Analysis is breaking down or breaking down the problem so that it is understood better. Aim is synthesizing or composition (putting together) the solution. These activities are now described in more detail.

5.3 Demands Exampleing with UML

In the Demands Example, the arrangement is considered as a black box. The Use Case Example is arisen in which the functional demands of the arrangement are defined in terms of use cases and actors. This section depicts the use of actors in real-time applications.

There are several variations on how actors are exampled [12,24,36]. An actor is very often a human user. In any data arrangements, humans are the only actors. It is also possible in data arrangements for an actor to be an external arrangement. In real-time and administered applications, an actor can also be an external INPUT/OUTPUT device or a timer. External INPUT/OUTPUT twists and timer actors are especially prevalent in real-time embedded arrangements, where the arrangement acts with the external natural world through sensors and actuators.

A human actor may use various INPUT/OUTPUT devices to physically act with the arrangement. In such cases, the human is the actor and the INPUT/OUTPUT devices are not actors. In some case, however, it is potential for an actor

to be an INPUT/OUTPUT device. This can encounter when a use case does not involve a human, as often happen in real-time applications.

An actor can also be a timer that sporadically broadcasts timer events to the arrangement. Periodic use cases are demanded when certain data demands to be output by the arrangement regularly. This is especially significant in real-time placements, although it can also be practicable in data placements. Although, some method actingologists consider timers to be internal to the placement, it is more useful in real-time application aim to conceive timers as logically external to the placement and to treat them as primary actors that initiate actions in the placement.

An example of a use case example from the Pump Control Arrangement is given in Figure 1, in which there are two use cases, Control Pump and View Pump Position. There are five actors, three constituting the three external sensors, one clock actor, and an external user actor, the Operator.

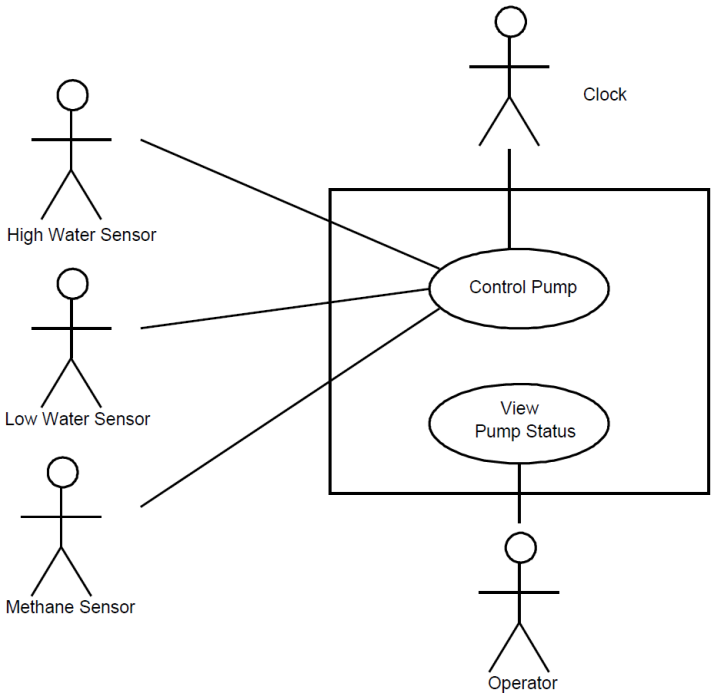

Figure1 Use case example for Pump Control Placement

5.4 Analysis Exampling with UML

This section depicts some of the interesting aspects of COMET for analysis exampling. In particular, this section depicts static exampling of the arrangement context, stereotypes to constitute object structuring decisions made by the analyst, and consistence checking between multiple views of a active example.

5.4.1 Static Exampling

For real-time applications, it is especially substantial to realize the port between the placement and the external environment, which referred to as the arrangement context. In Structured Analysis [42], the arrangement context is shown on an arrangement context plot. The UML notation does not explicitly support an arrangement context plot. However, the arrangement context may be described using either a static example or a communicating example [10]. An arrangement context class plot provides a more detailed view of the arrangement boundary than a use case plot.

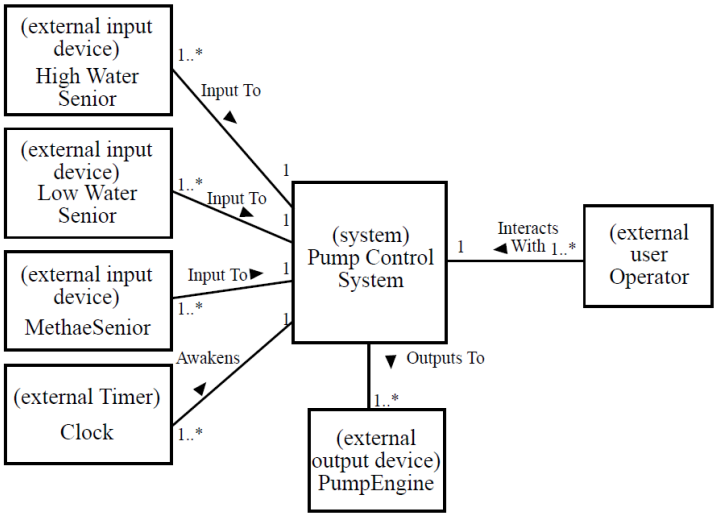

Figure 2 Pump Control Arrangement class context plot

Employing the UML annotation for the static example, the arrangement context is described demonstrating the arrangement as an aggregate class with the stereotype << arrangement >> and the external natural world is described as external classes to which the arrangement has to port. External classes are categorized using stereotypes. An external class can be an << external input device >> an << external output device >> an << external INPUT/OUTPUT device >> an << external user >> an << external arrangement >> or an << external timer >>. For a real-time arrangement, it is desirable to identify low level external classes that correspond to the physical INPUT/OUTPUT devices to which the arrangement has to port. These external classes are described with the stereotype << external INPUT/OUTPUT device >>. An example of an arrangement context class plot from the Pump Control Arrangement is given in Figure 2. There are three external input device classes, namely the three sensors, one external output device class, Pump Engine, one external timer class, and one external user class. For each external class, there is a one-to-many association with the Pump Control Arrangement. During the analysis exampling phase, static exampling is also used for exampling data-intensive classes [30].

5.4.2 Object Coordinating

Object coordinating criteria are allowed for to assist the architect in structuring an arrangement into objects. Several object-based and object-oriented analytic thinking method acting's provide criteria for deciding objects in the problem domain [4,9,14,24,28,43]. The COMET object structuring standards build on these method actings.

In object structuring, the destination is to classify objects in order that group together objects with like features. Whereas categorization based on inheritance is an objective of object-oriented exampling, it is fundamentally tactical in nature. Classification, however, is a strategic categorization. It is a decision to coordinate classes into certain groups because most package arrangements have these kinds of classes and classifying classes in this way helps us understand the arrangement we are to acquire.

UML pigeonholes are wont to differentiate among the different kinds of application program classes. A stereotype is a subclass of an existing exampling component, in this case an application class, which is wont to constitute a usage distinction, in this case the sort of class. A stereotype is described in guillemets, e.g., << port >>. An instance of a stereotype class is a stereotype object, which can also be demonstrated in guillemets. Thus an application class can be classified as an << entity >> class, which is a lasting class that stores data, an << port >> class, which port to the extraneous environment, a << control >> class, which furnishes the overall classification for the objects that take part in a use case, or an << application logic >> class, which capsulate algorithms separately from the data being controlled.

Real-time arrangements will have many device user port classes to port to the various detectors and actuators. They will also have composite state dependent control classes because these arrangements are highly state dependent.

5.4.3 Active Exampling

For concurrent, distributed, and real-time applications, active exampling is of particular grandness. UML does not stress consistency assuring between multiple views of the various cases. Nevertheless, during active exampling, it is substantial to realize how the finite state machine example, described using a state graph [19,21,22] that is accomplished by a state dependent control object, relates to the interaction example, which depicts the interaction of this object with other objects.

State Dependent Active Analysis accosts the interaction among objects that take part in state dependent use cases. A state dependant use case has a state dependent control object, which accomplishes a state graph, furnishing the overall control and determining of the use case. The interaction among the objects that take part in the use case is described on a communicating plot or sequence plot.

The state graph needs to be conceived in conjunction with the communicating plot. In particular, it is requirement to consider the messages that are experienced and sent by the control object, which accomplishes the state graph. An input event into the control object on the communicating plot must be consistent with the same event described on the state graph. The output event (which causes an action, enable or disable activity) on the state graph must be coherent with the output event shown on the communicating plot.

A case of the communicating plot for the Control Pump use case is given in Figure 3 and of the state chart for the Pump Control object is shown in Figure 4. In Figure 3, there are two input device port objects, High Water Sensor Port and Low Water Sensor User port, which receive inputs from the external input devices. There is one output device port object, Pump Engine Port, which outputs to the external output device. There is one state dependent control object, Pump Control, which executes the state graph in Figure 4. Finally, there is one timer object. Message inputs to the Pump Control object, such as High Water Discovered, in Figure 3 are the events that cause state changes on the state graph in Figure 4. Actions in Figure 4, such as Start Pump and Stop Pump, correspond to output messages from the Pump Control object in Figure 3.

5.5 Aim Exampling

This section depicts some of the interesting aspects of COMET for aim exampling. In particular, this section depicts the integration of communicating plots to synthesize an initial package aim, sub arrangement structuring using packages, distributed application aim, coincident job aim, and the aim of connectors using monitors.

5.5.1 The Conversion from Analytic thinking to Aim

In order to transition from analysis to aim, it is necessary to synthesize an initial package aim from the analysis carried out so far. In the analysis example, a communicating plot is arisen for each use case. The integrated communicating plot is a synthesis of all the communicating plots arose to support the use cases. The integration executed at this stage is analogous to the robustness analysis performed in other method actings [24,29]. These other method

actings use the static example for robustness analysis, whereas COMET stresses the active example, as this addresses the message communicating ports, which is crucial in the aim of real-time and distributed applications.

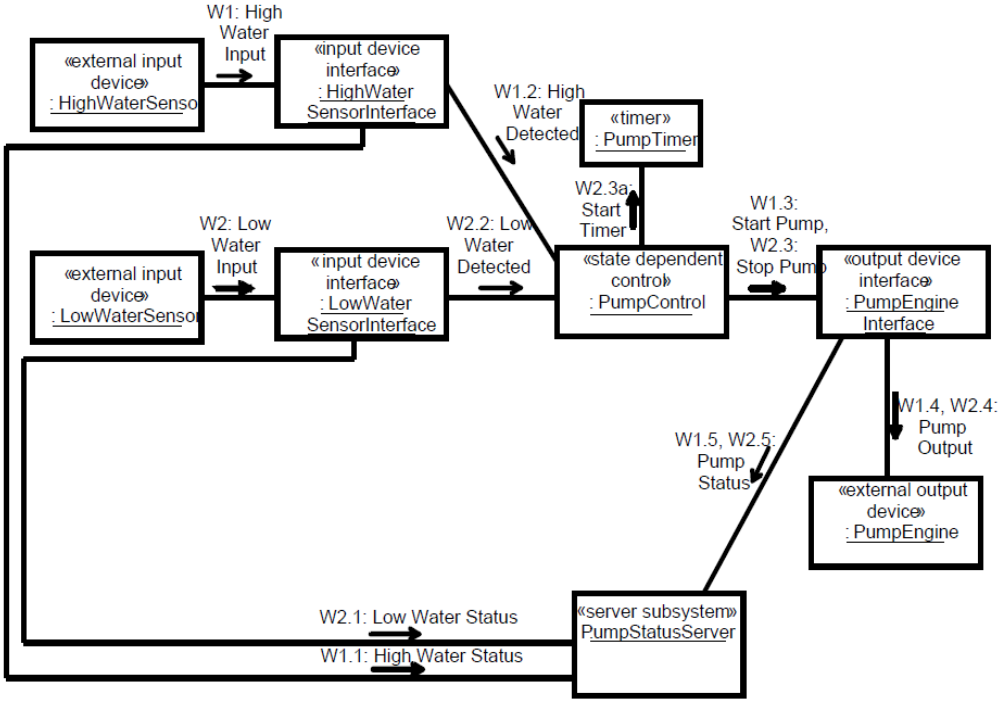

Figure 3 communicating plot for Control Pump use case

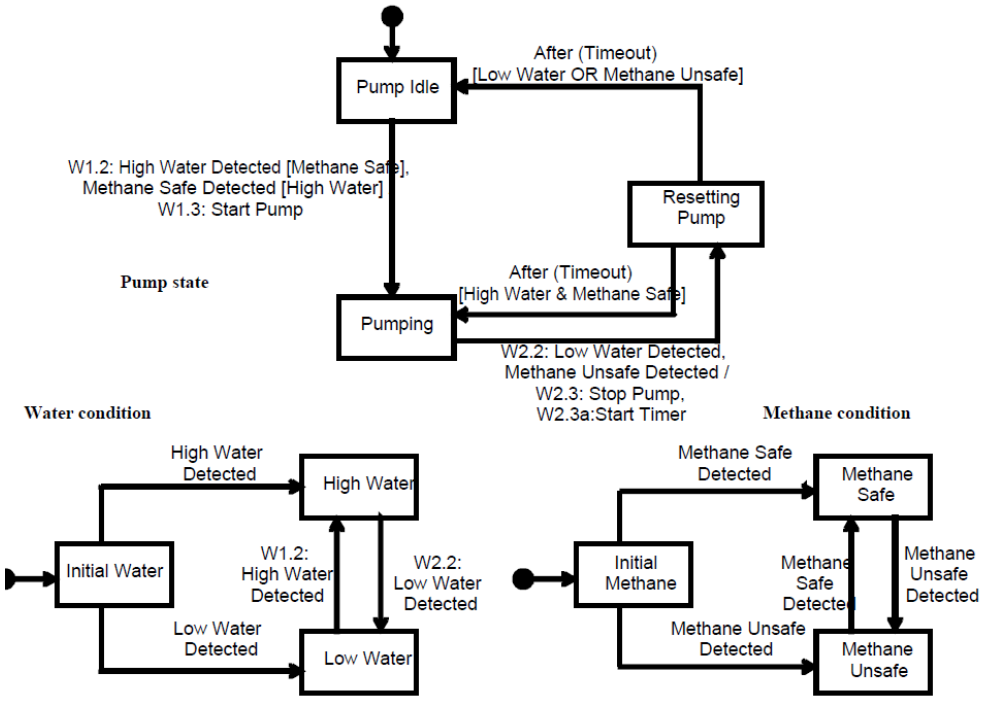

Figure 4 Pump Control state graphs

The incorporated communicating plot, which depicts the objects and messages from all the use case based, communicating plots, can get very large for a large arrangement and thus it may not be practical to show all the objects on one plot. This problem is addressed by developing integrated communicating plots for each sub arrangement, and developing a higher level sub arrangement communicating plot to show the active interactions between sub arrangements on a sub arrangement communicating plot, which depicts the overall package architecture, as shown in Figure 5. The structure of an individual sub arrangement is then described on an integrated communicating plot, which shows all the objects in the sub arrangement and their interconnections.

Figure 5 demonstrates three sub arrangements, Pump Sub arrangement (a control sub arrangement) of which there is many instances, Pump Status Server (a server sub arrangement) of which there is one instance, and Operator Port (a user port sub arrangement) of which there are many instances.

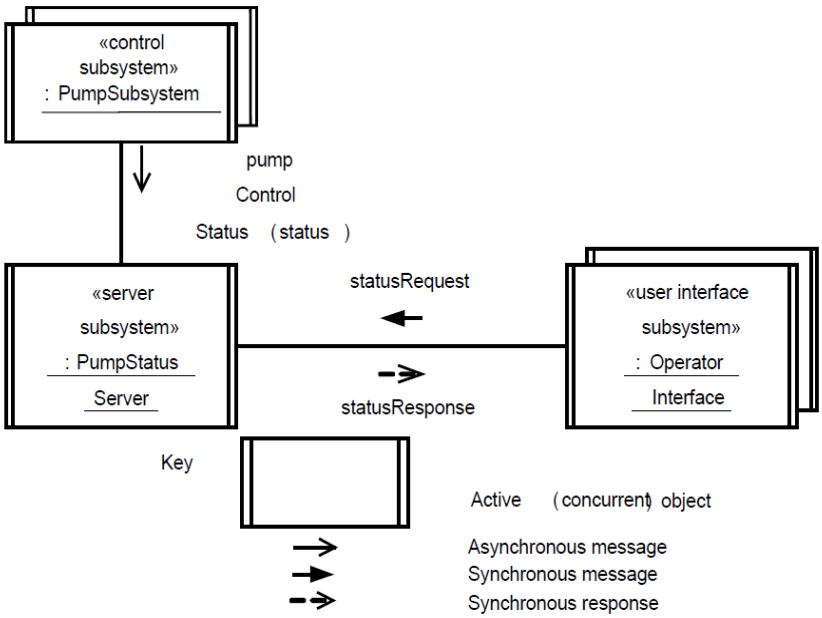

Figure 5 Administered package architecture

5.5.2 Package Architecture Aim

During Package Architecture Aim, the arrangement is broke down into sub arrangements and the ports among the sub arrangements are defined [33]. An arrangement is integrated into sub arrangements, which comprise objects that are functionally contingent on each other. The goal is to have objects with high coupling among each other in the same sub arrangement, while objects that are decrepit coupled are in different sub arrangements. A sub arrangement can be considered a composite or aggregate object that contains the simple objects that compose that sub arrangement.

5.5.3 Concurrent Communicating Plots

In the UML 2.0 notation, an active object or job is described as a box with two parallel lines on the left and right sides of the object box. An active object has its own thread of control and accomplishes concurrently with other objects. This is in contract to a passive object, which does not have a thread of control. A passive object only accomplishes when another object (active or passive) invokes one of its operations. In this paper, we refer to an active object as a job and a passive object as an object. Jobs are depicted on concurrent communicating plots, which depict the concurrency aspects of the arrangement [11]. A concurrent communicating plot depicts jobs with the active object notation described above and passive objects as ordinary boxes without parallel lines on the sides. In addition, decisions are made about the type of message communicating between jobs, asynchronous or synchronous, with or without reply.

5.5.4 Architectural Aim of Distributed Real-Time Arrangements

Distributed real-time arrangements accomplish on geographically administered nodes affirmed by a local or wide area network. With COMET, a distributed real-time arrangement is structured into distributed sub arrangements, where a sub arrangement is aimed as a assemble component and corresponds to a logical node. A sub arrangement component is defined as a accumulation of concurrent jobs executing on one logical node. As component sub arrangements potentially reside on different nodes, all communicating between component sub arrangements must be restricted to message communicating. Jobs in different sub arrangements may communicate with each other using several different types of message communicating (Figure 5) including asynchronous communicating, synchronous communicating, client/server communicating, group communicating, brokered communicating, and negotiated communicating.

In Figure 5, the Pump Sub arrangement conveys with the Pump Status Server using asynchronous communicating, while the Operator Port Sub arrangement communes with the Pump Status Server employing synchronous communicating with reply. The configuration of the distributed real-time arrangement is described on a deployment plot, as shown in Figure 6, which shows the three sub arrangements described as distributed nodes in a distributed configuration.

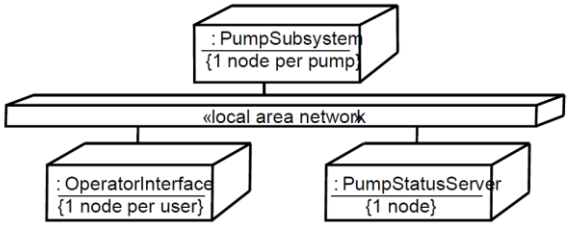

Figure 6 Distributed arrangement configurations

5.5.5 Job Coordinating

During the job coordinating phase, each sub arrangement is integrated into concurrent jobs and the job ports are defined. Job structuring criteria are provided to assist in mapping an object-oriented analysis example of the arrangement

to a coincident jobbing architecture. Following the approach used for object structuring, stereotypes are used to depict the different kinds of jobs. Stereotypes are also used to depict the different kinds of devices the jobs port to. During job structuring, if an object in the analysis example is determined to be active, then it is categorized further to show its job characteristics. For example, an active << INPUT/OUTPUT device port >> object is considered a job and categorized as one of the following: an << asynchronous INPUT/OUTPUT device port >> job, a << periodic INPUT/OUTPUT device port >> job, a << passive INPUT/OUTPUT device port >> job, or a << resource monitor >> job. Similarly an << external input device >> is classified, depending on its characteristics, into an << asynchronous input device >> or << passive input device >>.

An anachronistic Input/output device port job is demanded when there is an anachronistic INPUT/OUTPUT device to which the arrangement has to port. For each asynchronous INPUT/OUTPUT device, there needs to be an asynchronous INPUT/OUTPUT device port job to port to it. The asynchronous INPUT/OUTPUT device port job is actuated by an interrupt from the asynchronous device.

While an asynchronous INPUT/OUTPUT device port job deals with an asynchronous INPUT/OUTPUT device, a periodic INPUT/OUTPUT device port job deals with a passive INPUT/OUTPUT device, where the device is polled on a regular basis. In this situation, the activation of the job is periodic but its function is INPUT/OUTPUT related. The periodic INPUT/OUTPUT device port job is actuated by a timer event, performs an INPUT/OUTPUT operation, and then waits for the next timer event.

An example of job architecture for the Pump Sub arrangement of the Pump Control Arrangement is given in Figure 7. There are four jobs in the Pump Sub arrangement. There is one periodic input device port job, Methane Sensor Port, a temporal clustering job, Water Sensors, a control clustering job, Pump Controller, and a periodic job, Pump Timer.

5.5.6 Detailed Package Aim

In this step, the internals of composite jobs that arresting nested objects are aimed, detailed job synchronization consequences are addressed, connector classes are aimed that encapsulate the details of inter-job communicating, and each job's internal event sequencing logic is determined.

An example of the detailed aim of a composite job is given in Figure 8. The Water Sensors job of Figure 8 is a composite job that contains three objects, Water Sensors Coordinator, High Water Sensor Port and Low Water Sensor Port.

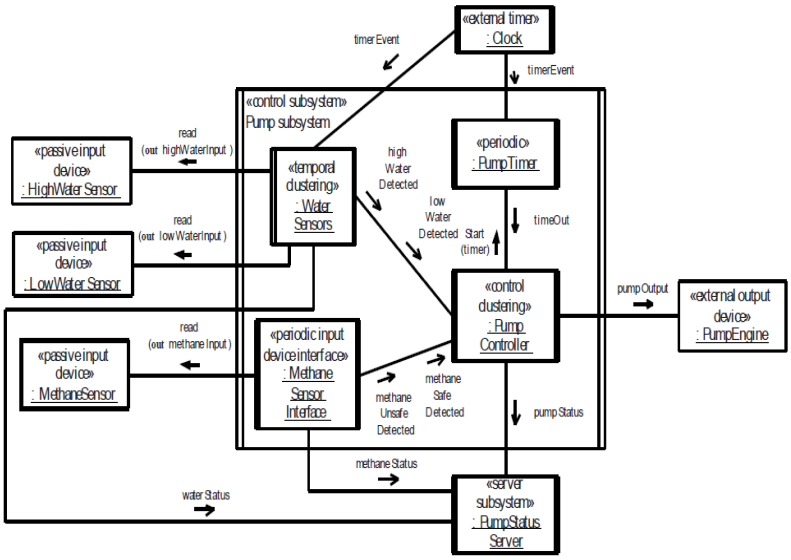

Figure 7 Pump Subarrangement - job architecture

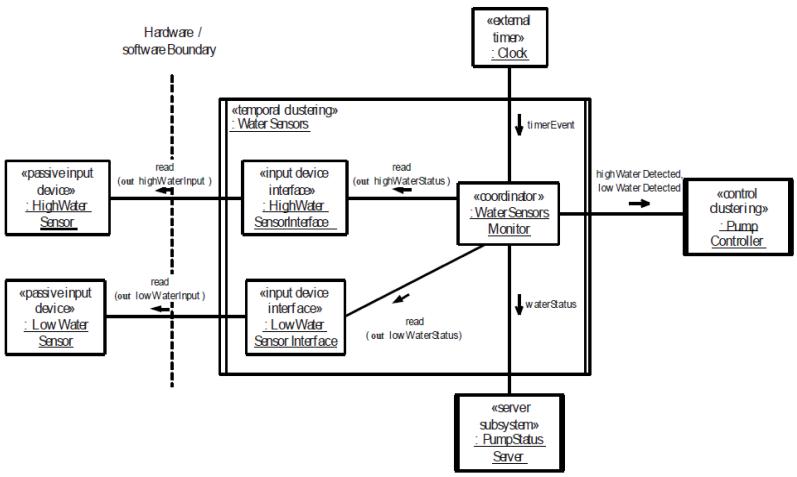

Figure 8 Water Sensors - secular clustering with nested device port objects

If a passive class is got at by more than one job, then the class's functioning's must synchronize the access to the data it encapsulates. Synchronization is accomplished using the mutual exclusion or multiple readers and writers algorithms [2].

Connector classes capsulate the details of inter-job communicating, such as loosely and tightly coupled message communicating. Some concurrent programming languages such as Ada and Java allow for mechanisms for inter-job communicating and synchronization. Neither of these languages supports approximately coupled message communicating. In order to allow for this capability, it is necessary to aim a Message Queue connector class, which encapsulates a message queue and allows for operations to access the queue. A connector is aimed using a monitor, which aggregates the concepts of data hiding and job synchronization [2,26]. These monitors are used in a single processor or multiprocessor arrangement with shared memory. Connectors may be aimed to handle loosely coupled message communicating, tightly coupled message communicating without reply, and tightly coupled message communicating with reply.

VI. PACKAGE ARCHITECTURAL PATTERNS FOR REAL-TIME CONTROL

Package architectural patterns [7,16] allow for the skeleton or template for the overall package architecture or high-level aim of an application. Basing the package architecture of a product line on one or more package architectural forms assistances in aiming the original architecture as well as evolving the architecture.

There are two main classes of package architectural patterns [16]. Architectural structure patterns address the static structure of the package architecture. Architectural communicating patterns address the message communicating among distributed components of the package architecture.

Most package arrangements can be based on well realized overall package architectures. For example, the client/server package architecture is prevalent in many package applications. There is the basic client/server architecture, with one server and many clients. However, there are also many variations on this theme, such as multiple client / multiple server architectures and brokered client/server architectures.

Many real-time arrangements allow for overall control of the environment by providing either centralized control, decentralized control, or hierarchical control. Each of these control approaches can be exampled using a package architectural pattern. In a centralized control pattern, there is one control component, which executes a state graph. It receives sensor input from input components and controls the external environment via output components, as shown in Figure 7 for the Pump Controller job. In a centralized control pattern, the control component executes a state graph, which for Pump Controller is described in Figure 4. Another pattern used in the Pump Control Arrangement is the client/server pattern, as shown in Figure 5, where the Pump Sub arrangement and Operator Port Sub arrangement are the clients and the Pump Status Server is the server.

Architectural communicating patterns for real-time arrangements include asynchronous communicating and synchronous communicating, both with and without reply. Other possible communicating patterns include subscription/notification patterns and broker patterns. In the Pump Control Arrangement, both asynchronous and synchronous message communicating are used as shown in Figures 5 and 7.

VII. FUNCTIONING ANALYSIS OF REAL-TIME AIMS

Functioning analysis of package aims is particularly significant for real-time arrangements. The effects of a real-time arrangement failing to meet a deadline can be catastrophic.

The quantitative analysis of a real-time arrangement aim allows the early detection of potential Functioning problems. The analysis is for the package aim conceptually executing on a given hardware configuration with a given external workload applied to it. Early detection of potential Functioning problems allows alternative package aims and hardware forms to be inquired.

In COMET, Functioning analysis of package aims is achieved by applying real-time scheduling theory. Real-time scheduling is an approach that is particularly appropriate for hard real time arrangements that have deadlines that must be met [14,37]. With this approach, the real time aim is analyzed to determine whether it can meet its deadlines.

A second access for examining the Functioning of an aim is to use event sequence analysis and to integrate this with the real-time scheduling theory. Event sequence analysis considers scenarios of job collaborations and annotates them with the timing parameters for each of the jobs participating in each collaboration, in addition to arrangement overhead for inter-object communicating and context switching [15]. The equivalent period for the active objects in the collaboration is the minimum inter-arrival time of the external event that initiates the collaboration.

VIII. REAL-TIME EMBEDDED PACKAGE PRODUCT LINE AIM

A package product line (SPL) comprises of a family of package arrangements that have some common practicality and some variable functionality [27,8,16]. Package product line directing involves developing the requirements, architecture, and component implementations for a family of arrangements, from which products (family members) are derived and configured. The problems of developing individual package arrangements are scaled upwards when developing package product lines because of the increased complexity due to variability management.

A better agreement of an arrangement or product line can be found by conceiving the multiple views, such as requirements examples, static examples, and active examples of the arrangement or product line. A graphical exampling language such as UML helps in developing, understanding and communicating the different views. A key view in the multiple views of a package product line is the feature exampling view. The feature example is crucial for managing

variability and product derivation as it depicts the product line requirements in terms of commonality and variability, as well as defining the product line dependencies. Furthermore, it is necessary to have a growth approach that promotes package evolution, such that original growth and subsequent maintenance are both treated using feature-driven evolution.

The Evolutionary Package Product Line Engineering Process [16] is a highly iterative package process that eliminates the traditional distinction between package growth and maintenance. Furthermore, because new package arrangements are outgrowths of existing ones, the process takes a package product line perspective;

it consists of two main processes, as shown in Figure 9:

a) Product line Engineering. A product line multiple-view example, which addresses the multiple views of a package product line, is developed. The product line multiple view example, product line architecture, and reusable components are developed and stored in the product line reuse library.
b) Package Application Engineering. A package application multiple-view example is an individual product line member derived from the package product line multiple view example. The user selects the required features for the individual product line member. Given the features, the product line example and architecture are adapted and tailored to derive the application architecture. The architecture determines which of the reusable constituents are needed for configuring the executable application.

Package product line concepts can also be applied to the aim of embedded real-time package. Thus the COMET aim method acting has been extended to the PLUS method acting (Product Line UML-based Package engineering) for aiming embedded real-time package product lines as described in [16].

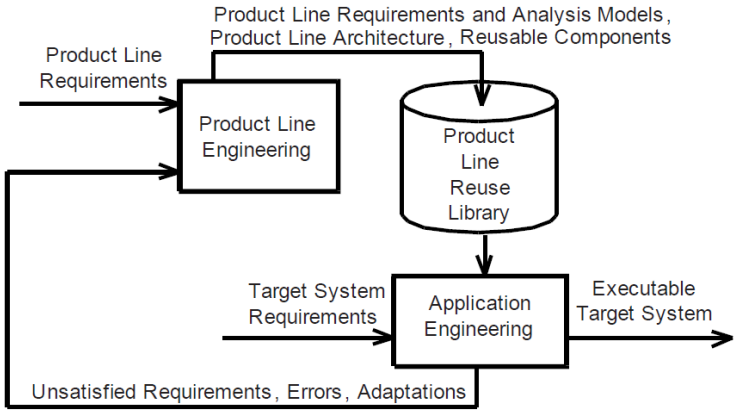

Figure 9 Process examples for package product line engineering

IX. CONCLUSIONS

This paper has described concepts and method actings for the aim of real-time embedded package arrangements. When aiming concurrent and real-time arrangements, it is necessity to blend object-oriented concepts with the concepts of concurrent processing. This paper has given an overview of the COMET example-based package aim method acting for aiming real-time embedded arrangements, which integrates object-oriented and concurrent processing concepts and uses the UML notation. With the proliferation of low cost workstations and personal computers operating in a networked environment, the interest in aiming concurrent arrangements, particularly real-time and distributed arrangements, is likely to grow rapidly in the next few years. Furthermore, with the growing need for reusable aims, aim method actings for package product lines are likely to be of increasing importance for future real-time embedded package arrangements.

REFERENCES

[1] M. Awad, J. Kuusela, and J. Ziegler, “Object-Oriented Technology for Real-Time Arrangements”, Prentice Hall, 1996.
[2] Bacon J., “Concurrent Arrangements”, Third Edition, Addison Wesley, Boston MA, 2003.
[3] Barnes, J., “Programming in Ada 95”, Addison Wesley, Boston MA, 1995.
[4] Booch G. et. al., "Object-Oriented Analysis and Aim with Applications", Addison Wesley, Boston MA, Third Edition, 2007.
[5] G. Booch, J. Rumbaugh, I. Jacobson, “The Unified Exampleing Language User Guide”, Addison Wesley, Boston MA, Second Edition, 2005.
[6] R.J.A. Buhr and R.S. Casselman, “Use Case Maps for Object-Oriented Arrangements”, Prentice Hall, 1996.
[7] F. Buschmann et al, Pattern-Oriented Package Architecture: A Arrangement of Patterns, John Wiley & Sons, 1996.
[8] P. Clements and L. Northrop, Package Product Lines: Practices and Patterns, Addison Wesley, 2002.
[9] Coad P and E Yourdon, "Object-Oriented Analysis", Prentice Hall 1991.
[10] B. P. Douglass, “Doing Hard Time: UML, Objects, Frameworks, and Patterns in Real-Time Package Growth”, Addison Wesley, Boston MA, 1999.
[11] B. P. Douglass, “Real-Time UML”, Third Edition, Addison Wesley, Boston MA, 2004.
[12] Fowler M and K. Scott, "UML Distilled", Third Edition, Addison Wesley, Boston MA, 2004.

[13] H. Gomaa, "A Package Aim Method acting for Real Time Arrangements", Communicatings ACM, Vol. 27, No. 9, September 1984.
[14] H. Gomaa, Package Aim Method actings for Concurrent and Real-Time Arrangements, Addison Wesley, Boston MA, 1993.
[15] H. Gomaa, "Aiming Concurrent, Distributed, and Real-Time Applications with UML", Addison Wesley, Boston MA, 2000.
[16] H. Gomaa, "Aiming Package Product Lines with UML: From Use Cases to Pattern- based Package Architectures", Addison Wesley, Boston MA, 2005.
[17] H. Gomaa, "A Package Exampleing Odyssey: Aiming Evolutionary Architecture-centric. Real-Time Arrangements and Product Lines", Keynote paper, Proc. ACM/IEEE 9th International Conference on Example-Driven Engineering, Languages and Arrangements, Springer Verlag LNCS 4199, Pages 1-15, Genova, Italy, October 2006.
[18] H. Gomaa and M.E. Shin, "A Multiple-View Meta-Exampleing Approach for Variability Management in Package Product Lines", Proc. International Conference on Package Reuse, Madrid, Spain, Springer LNCS 3107, July 2004.
[19] Harel, D., "On Visual Formalisms." CACM 31, 5 (May 1988), 514-530.
[20] Harel, D. and E. Gary, "Executable Object Exampleing with Stategraphs", Proc. 18th International Conference on Package Engineering, Berlin, March 1996.
[21] Harel, D. and M. Politi, "Exampleing Reactive Arrangements with Stategraphs", McGraw-Hill, 1998.
[22] Hatley D. and I. Pirbhai, "Strategies for Real Time Arrangement Specification", New York: Dorset House, 1988.
[23] Jackson M., "Arrangement Growth", Prentice Hall, 1983.
[24] I. Jacobson, Object-Oriented Package Engineering, Addison Wesley, Boston MA, 1992.
[25] I. Jacobson, G. Booch, J. Rumbaugh, "The Unified Package Growth Process", Addison Wesley, Boston MA, 1999.
[26] J. Magee and J. Kramer, "Concurrency: State Examples & Java Programs", John Wiley & Sons, Second Edition, 2006.
[27] Parnas D., "Aiming Package for Ease of Extension and Contraction",IEEE Transactions on Package Engineering, March 1979.

