A Completely Integrated Approach to Developing, Implementing, Evaluating Distributed Active Database Management and its OS Support

Horst F. Wedde* Kwei-Jay Lin**
Aloysius K. Mok* Krithivasan Ramamritham******

* University of Dortmund, Germany
**UC Irvine, USA
***University of Texas, Austin, TX, USA
****IIT Bombay, India & UMass, Amherst, Mass., USA

Abstract: This paper constitutes a work-in-progress report on the first, mostly conceptual phase of a major international effort in building and evaluating a distributed testbed for database application systems in safety-critical real-time environments. Given that safety/ reliability requirements and real-time constraints are in conflict there cannot be a closed form solution or design, and all system functions have to dynamically adapt to the unpredictable environmental situation in safety-critical real-time systems. Over the past years groups headed by the authors had pursued novel concepts and approaches on a novel tailored LINUX kernel, safety-critical application operating system support (MELODY project) and Active Database levels (concurrency control under data replication). In the research project a complete integration of these achievements from the basic OS kernel through the application levels is the major theme. While quite a number of research and design problems stemming from the partial insights or incomplete functionality on the various levels posed serious challenges the integration itself requires, and gives rise to, extensions, modifications, or refinements of the functions involved for ensuring system survivability. The paper will describe the technical details of these implications from the integration for the whole project as well as the stepwise design and evaluation of the different functions and models.- To our knowledge this is the first fully integrative attempts in the area of distributed safety-critical real-time systems.

Keywords: distributed operating systems, real-time systems, safety-critical systems, concurrency control, active databases, similarity
Address for correspondence: Prof. Dr. Horst F. Wedde, Informatik III, University of Dortmund, 44221 Dortmund / Germany, wedde@cs.uni-dortmund.de

1. TASKS IN SAFETY-CRITICAL SYSTEMS.

In *safety-critical systems,* (such as nuclear power plants, distributed cooperation of autonomous robots in Outer Space or on the plant floor, automated aircraft landing systems for bad weather conditions, etc.) tasks not only have to meet deadlines, but most are critical in the sense that the system would not survive in case of a task-specific number of deadline failures of subsequent task instances. In such a critical stage, an instance is said to have become ***essentially critical***. However, beyond this special real-time responsiveness (here successful handling essentially critical task instances which was termed ***survivability*** (Wedde, Lind(1997)), safety-critical systems must also satisfy rigid ***dependability*** requirements. A high amount of ***adaptability*** of system functions is demanded to meet these ***conflicting requirements***, since the unpredictable environment does not allow for a static trade-off between these classes of restrictions.

Organization of the Paper. The theme of the paper is a multi-level integration of research work in real-time operating system kernels, operating system support for safety-critical applications, and real-time adaptable transaction processing. In order to substantiate, and further elaborate and expand on, the concepts addressed a novel application scenario will be described in section 2. Section 3 gives a brief survey over the novel MELODY operating systems functions. At the same time, it is an introduction into our novel design and analysis methodology termed Incremental Experimentation which is central for the whole project. Section 4 is a short intimation of our transaction model and its refinement through criticality and sensitivity. In section 5 we sketch a similarity concept studied previously. We indicate its use for refining both MELODY functions (File Server and File Assigner) and the concurrency control algorithms described in section 4, also pointing to the potential benefits for system survivability in the integrated model.

2. AUTOMATED LANDING SYSTEMS

Corrective actions in automated aircraft landing systems (ALS) contribute to adjustments for ap-

proach and landing of an aircraft. The environments of these systems are categorized regarding the ability of the pilots to assume control of the aircraft. Category 1 conditions are those in which the line of sight could be made) of 200 meters. Category 2 conditions reduce the line of sight to 400 meters which results in a decision height of only 100 meters. In category 3 conditions the line of sight is further reduced to 200 meters which effectively reduces the decision height to 0. Typically, all aircraft are grounded (even military aircraft) in category 3 weather conditions in order to avoid the loss of lives.

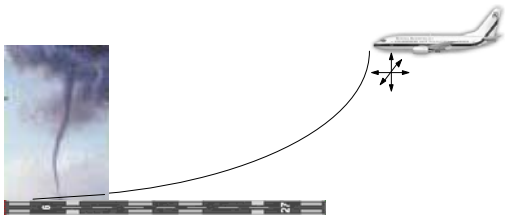

Fig.1: Bad Weather Landing

Automated landing systems could help to allow flights to land even during category 3 weather conditions such as depicted in fig.1 . Corrective actions to bring the aircraft back on course, or keep it there, are computed by tasks which use information periodically gathered by sensors at the aircraft periphery and filtered/ structured by distributed preprocessors. The response times of the mechanical control systems are considerably long (up to more than 4 sec). During this time the environmental situation may have changed drastically which may render a correction obsolete, or even dangerous, for the safety of the plane. Therefore such actions have to be designed to cause considerably *small-scale changes* which, in the negative case mentioned, are unlikely to have disastrous consequences. The computing tasks would be performed whenever necessary (typically aperiodically).

A key idea is that in a series of small-scale corrections occurring with a high frequency, each of them may outbalance the effect of the previous ones while contributing to gradually forcing the aircraft back on course. Also the information on the environment would be more frequently taken into account resulting in more up-to-date reactions. For this purpose the deadlines of the corresponding task instances are set very tight. When a task instance is about to miss its deadline it will be aborted since we assume subsequent task instances may make up for the previous ones to a certain extent. However, as the aircraft may meanwhile get considerably off its course it becomes more and more critical to successfully complete a correction in due time. These actions, while then potentially more and more rigid, might eventually cause the plane to be destabilized. Therefore, for every corrective activity, the tasks are to be designed such that

(2) there is a preset number of tolerable consecutive deadline failures of task instances. Beyond this the next deadline is *hard* or *essentially critical* is at least 600 meters and results in a decision height (maximum height at which a decision to abort landing

(see section 1).

(3) whenever during this series of instances a deadline has been met – the associated corrective action being assumed to have the outbalancing effect mentioned – it has the same frame of tolerance as the previous successful instance.

We have formalized this into a concept of ***(relative) task criticality*** (Wedde, Lind(1997)). The more critical it is for a corrective action to be performed in due time the more tolerable it is for the survivability of the system to trade up-to-date information – which may not be available in time in our type of distributed systems – for information that is *nearly up-to-date* while available at the computer site where it is needed (by a reading task). So, beyond a preset number of consecutive deadline failures the use of nearly up-to-date information is considered satisfactory. This has been formalized into a concept of ***(relative) task sensitivity*** (Wedde, Lind(1997)).

3. THE ADAPTABLE SAFETY-CRITICAL OPERATING SYSTEM MELODY

Task criticality and sensitivity are crucial operating system features in the MELODY project to guarantee survivability of the application. We will explain MELODY's major functional novelties in this section.

As both the adaptability and the conflicting constraints of real-time responsiveness and dependability were to be respected in MELODY (see section 1) we started the project development with a rather simple model for studying novel adaptive file system functionality ***(phase 1***) in order to cope with this enormous complexity. In order to provide for a variable file replication a novel function, the ***File Assigner (FA),*** was created. Not only was the number and location of mutually consistent file copies (termed ***public copies)*** to be arranged according to the needs of local tasks. There was also a relaxed distributed protocol introduced by which certain copies would be refreshed after each update of a file *(i.e. of its public copies)* was completed. The latter copies were called ***private copies.*** They represent nearly up-to-date information (see section 2) as long as there were only gradual changes to the file. According to the varying weight of the real-time and dependability constraints the local File Assigners executed a distributed consensus protocol to manage the number, location, and quality of file copies, at the same time reflecting that read tasks (in phase 1!) would use public or private copy information interchangeably, depending on local availability only. (Write tasks write to the public copies only, by definition.)

Since there is no overhead for maintaining consistency among private copies a model which provides

for private copy replication only (one public copy only) ***(Private Copy Model),*** could be expected to have clear performance advantages (regarding the number of deadline failures) over a model which allows for public copies only ***(Public Copy Model)*** *as long as there* is a high dominance of read tasks (which entails a very low number of public copy updates, hence very infrequent private copy refreshments. (Public copy maintenance requires a considerably higher overhead.) Conversely, under a high write dominance the frequent refreshments of the private copies (sending a fresh copy to the private copy site) would probably jam the network while the overhead for mutual consistency could be assumed to be approximately linear with the number of public copies and competing tasks.

This was investigated in comparative simulation experiments. Both models were, of course, compared with MELODY but also with two benchmark models: the ***Base Model*** (no FA) and the ***Ideal Case Model*** (FA costs set to zero). Some results are depicted in fig. 2 and 3. They clearly showed that the expectations mentioned for the Private and Public Copy models were valid. Even more importantly, MELODY outperformed both of them throughout all simulation experiments. Its higher overhead (compared to the simpler models) was more than outweighed by its higher flexibility.

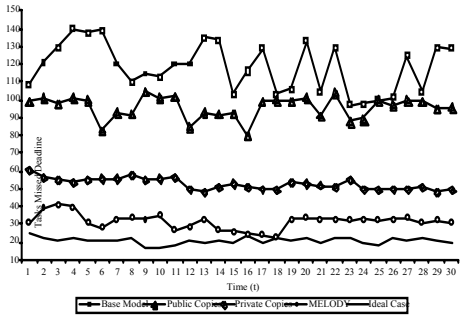

Figure 2: Read Dominance Task Profile

With these results in mind the MELODY task model was refined by including criticality and sensitivity (see section 2). The FA functions had e.g. to be modified in that sensitive read task instances would *always* require public copy information. This constituted ***phases 2 & 3*** of MELODY. Technical details will otherwise have to be omitted here, due to page limitations. They can be found in (Wedde, Lind(1997)).

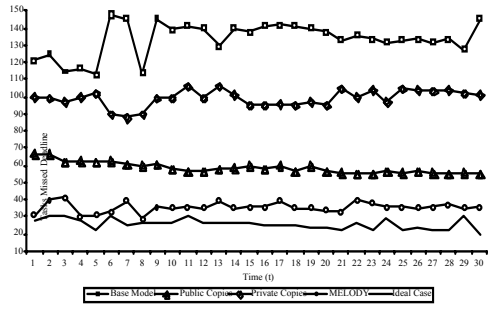

Figure 3: Write Dominance Task Profile

In order to connect the insights from phase 1 to the refined model, extensive simulation was done such that the task profiles chosen for the new MELODY stage were solely consisting of non-critical and non-sensitive *(robust)* tasks. Our expectation was that the previous experimental results (see fig. 2,3) could (and should) be mimicked in the new MELODY model. This was the case throughout (Wedde, Lind(1997)). While in a second round criticality and sensitivity values were widely varied it turned out that with increased sensitivity the roles of the Public and Private Copy models were surprisingly flipped (fig. 4) thus revealing a strong influence of sensitivity on the deadline failure rate performance. Criticality, in turn, has a remarkable impact on the system *survivability* (ability of meeting all essentially critical or hard deadlines). Again, we refer to (Wedde, Lind(1997)) for technical details.

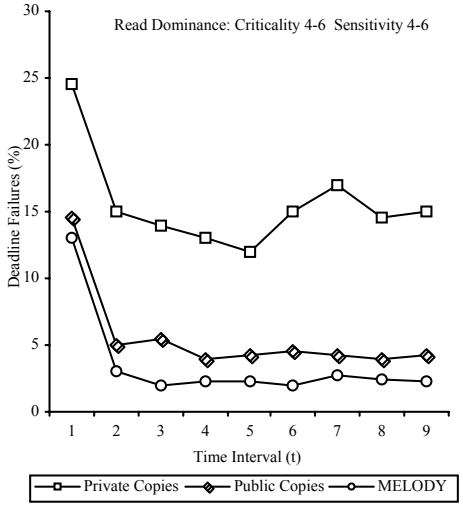

Figure 4: Read Dominance and Sensitive Tasks

These three MELODY development phases discussed represent the major features of our heuristic and experiment-driven design methodology called ***Incremental Experimentation.*** It leads to an iterative procedure of refinement and extension which follows a scheme of using experimental insights in one phase, leading to concepts and expectations for system refinement or extension in the subsequent phase, evaluating the previous insights within the refined/ extended model context, deriving new insights by utilizing the evaluation just mentioned etc. An example for the efficiency of our method is that the results of the impact of criticality and sensitivity on the system performance described for phases 2 & 3, could hardly have been achieved without doing phase 1 before building the refined model. So far, we have successfully competed completed six phases in the MELODY project.

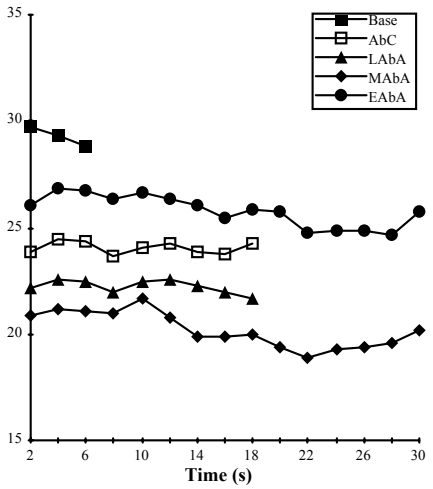

Figure 5: Medium Competition Task Profile

As phases 2 and 3 were being completed, novel distributed resource scheduling algorithms (Priority Insertion, Delayed Insertion and On-the-Fly protocols) had been defined and validated (through extensive comparative simulation experiments) in ***phase 4*** (Daniels(1992)). After a complete comparative evaluation the Delayed Insertion protocol was chosen to implement the distributed resource scheduling in MELODY.

Different from traditional operating system design the distributed task execution on distributed resources in MELODY led us to *reverse the order of task and resource scheduling* **(phase 5).** To make up for the inaccurate information the Task Schedulers were left with the ***Run-Time Monitors (RTM)*** were introduced which, at each site, would abort tasks as early as possible, based on precise information from the local FS. At the same time competing task instances at remote sites would benefit from resources being locked as late as possible in the task execution course at a given site. Technical details are in (Wedde et al.(1998)).

The local RTM was also put in charge of a ***dynamic integration of Task Scheduling and File Server activities.*** These measures proved to be very effective for MELODY. Details on the very extensive simulation studies can be found in (Wedde et al.(1998)).

In a second effort we investigated and refined the Task Monitoring function of RTM considerably. The idea behind was again to trade earliest possible task abortion (carrying a measurable decision inaccuracy) for an abortion based on accurate information but occurring possibly much later in the task life phases. The key for this approach is that wrong abortion decisions could be tolerated *to a certain degree* (see section 2). So, beyond the previous RTM function to abort before computation (AbC model) we provided for three more models in which abortion decisions would even be made during the initial part of the resource acquisition phase: ***the Latest Abort before Acquisition (LAbA), Medium Abort before Acquisition (MAbA),*** and ***Earliest Abort before Acquisition (EAbA).***

Since at this time all MELODY servers had been fully and explicitly integrated (Wedde et al.(1996), Wedde et al.(1998), Wedde et al.(1999b)) simulation did not appear any longer as an adequate evaluation method since we had no longer a sufficient understanding of the impact coming from pre-specified parameters like task execution, communication, resource acquisition times etc. This left us without a decent basis of judgment about the quality of our highly complex service and integration mechanisms. Therefore the TM evaluation, as well as the previous evaluations, was done through ***distributed experiments,*** in ***phase 6.*** (An in-depth technical discussion can be found in (Wedde et al.(1999b)) based on our own method of internal time synchronization (Wedde et al.(1999a)).

While the new protocols outperformed the AbC model throughout the MAbA model performed best looking over the whole range of task profiles, both under the deadline failure rate and survivability performance criteria, and was therefore adapted for MELODY (see figure 5). As a unique result, although the EAbA model was worse than both the AbC and LabA models regarding the deadline failure rate, it clearly survived its two competitors for both the medium and high competition profiles. While this appears counterintuitive at first glance it proves in fact that deadline failure performance and survivability are independent real-time measures.

4. DISTRIBUTED ACCESS TO REPLICATED DATA

Recently novel concurrency algorithms have been constructed for replicated data access under firm deadlines, and extensively evaluated under simulation (Xiong et al. (1999) The idea was to define a new Optimistic Distributed Concurrency Control protocol (OCC) and a new Distributed Optimistic Two-Phase Locking algorithm (O2PL). They operate on transactions modeled as depicted in fig.6. Assuming that preemption is used on the master level only a number of data conflict resolution mechanisms (Priority Blocking, Priority Abort, Priority Inheritance) and combinations are used to avoid, or minimize, priority inversion.

If we identify cohorts as tasks the MELODY task model includes the cohort model in fig.6 in which updaters at different sites take care of the replicas. (MELODY tasks executions are not limited to replication of updates but could be considered as small-scale transactions.) No provision is taken for adaptability in safety-critical environments. In the proposed international project, we will introduce criticality and sensitivity on the cohort level assuming that – like deadlines – these parameters may have different values (or otherwise all cohorts have the same value), thus creating adaptability to the unpredictable environmental requirements on the transaction level. The new protocols will be evaluated in *distributed experiments*. They will be compared to each other, and with their non-adaptable yet simpler versions, and with any other adaptable protocols (so far they will be available during this advanced project phase).

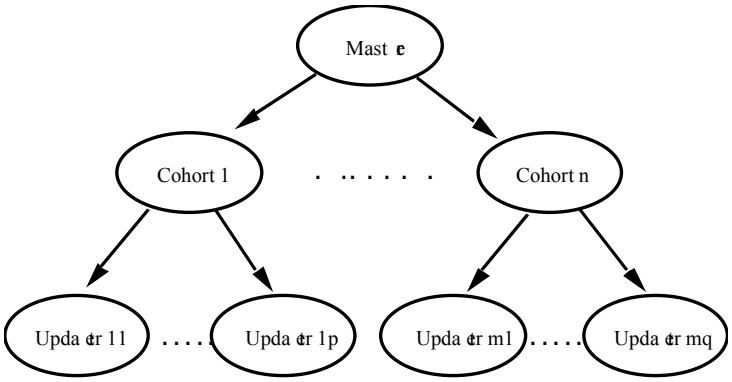

Figure 6: Transaction Model for Replicated Data References

5. DATA AND TASK SIMILARITY

In a distributed safety-critical system such as depicted in fig.1 there are sensors at the periphery of the application which input a continuous stream of data into preprocessors. Here periodic tasks filter, digitize, and structure these data. To each data object O originating from this process – which is a template for its instances – a ***similarity bound sm(O)*** will be assigned. (More technical details can be found in (Kuo et al.(1997)).)

(0) Note that this bound could be dynamically reset for different task instances (as e.g. the aircraft in fig.1 approaches a more critical phase of landing). Typically the bound would be lowered as the environment gets more unpredictable.

(1) Every task t_j will also be assigned a *similarity bound sm (t_j,O)* which is related to the effect of t_j using O for an execution instance. (In a first phase we will assume that all sm (t_j.O) are equal for the objects which t_j needs. An easy example is a cohort updating replicas of one object.) sm(t_j,O) will be sent to the site from which O originates. We will assume that

$sm(O) \leq \min\{sm(t_j) \mid t_j \text{ needs } O\}$.

(2) If for a data object O generated at a preprocessor P_i, or another processor, a subsequent instance is no longer similar to the previous one (as specified by sm(O)) – say it differs by a(O) > sm(O) - then the object manager at P_i will compare a(O) to sm(t_j) where t_j needs O. If a(O) ≤ sm(t_j) then tj would not be invoked. If a(O) > sm(t_j) then a message will be sent to the site holding t_j to invoke the next task instance.

(3) The condition under (1) makes sure that task instances will be invoked whenever necessary, i.e. whenever sm(t_j) is exceeded, since the alerting procedure at the site generating O would be invoked in time. At the same time this sets up, in an explicit model *for safety-critical systems and their environment,* the principle of *typically aperiodic tasks* as it underlays MELODY.

Note: Sensitivity and similarity are closely related concepts in the MELODY context: Both refer to the difference between earlier and later information. However, they are still independent parameters: First, sensitivity is no measure for invoking a task instance. Also, as the situation for the computer system becomes more and more unpredictable (and subsequent task instances would fail more and more frequently) the relative sensitivity will be more and more relaxed (see section 2) while similarity bounds get tighter instead (as was remarked in (0)). Another difference is that sensitivity directly relates to task management while similarity reflects the data management.

Similarity will be defined for tasks in MELODY as well as for cohorts in the transaction model introduced in fig.6. It will be utilized in the project to further modify and improve the adaptability on different levels:

OS level (MELODY): Upon an update of a file f, i.e. of the public copies of f (see section 3), if the change to f is within preset similarity bounds then the update operation which refreshes the private copies would not be exe-

cuted at this time. Also, when updated data objects are similar to the previous instance tasks which operate on these objects would not be invoked.

Database level: The concurrency algorithms mentioned in section 4 will be assigned similarity values for every cohort, or for the master (if the cohort values are set to be all equal).

All actions combined, the following project phases emerge:

- Creating an adaptable real-time LINUX version to handle the shell-level MELODY functions;
- Extending MELODY functions FA, FS by including similarity; realize similarity of files, tasks, according to an appropriate application study;
- Extending the transaction model by introducing criticality and sensitivity of cohorts or of the master;
- Extending the transaction model by introducing similarity of cohorts or of the master;
- Evaluating the different modified concurrency control protocols, at the same time comparing them with the less adaptable versions, in distributed experiments which are incrementally performed, according to the principles of Incremental Experimentation (see section 3)
- Investigating existing application studies, in particular in automated flight control.

This constitutes a mid-range research effort (5-6 years) involving four institutions from Europe, India, and the US. From initial studies we derived the hope that the more sophisticated models of our completely integrated computer system will significantly outperform the earlier system models, in particular regarding the most crucial measure: *survivability*.

6. REFERENCES

Daniels, D.C. (June 1992). The Design and Analysis of Protocols for Distributed Resource Scheduling under Real-Time Constraints. *Ph.D. Dissertation*; Wayne State University.

Kuo, T., S. Ho and A. Mok (December 1997). Similarity-Based Load Adjustment for Real-Time Data-Intensive Applications; Proc. of the 18th IEEE Real-Time Systems Symposium; San Francisco.

Xiong, M., K. Ramamritham, J. Haritsa and J.A. Stankovic (June,2, 1999). Regulating Concurrent Access to Replicated Data in Distributed Real-Time Systems. Proceedings of the IEEE Workshop on Dependable and Real-Time E-Commerce Systems (DARE'98), Denver, Colorado.

Wedde, H.F., and J.A. Lind (1997). Building Large, Complex, Distributed Safety-Critical. *Real-Time Systems* **Vol. 13 No. 3**

Wedde, H.F., C.Stange and J.A. Lind (November 1996). Integration of Adaptive File Assignment into Distributed Safety-Critical Systems. *WRTP'96, 21st IFAC/IFIP Workshop on REAL TIME PROGRAMMING*; Gramado, RS, Brazil.

Wedde, H.F., and J.A. Lind (1998). Integration of Task Scheduling and File Services in the Safety-Critical System MELODY. Proceedings of the EUROMICRO '98 Workshop on Real-Time Systems, Berlin, Germany.

Wedde, H.F., J.A. Lind and G. Segbert (June 1999). Achieving Internal Synchronization Accuracy of 30 μs Under Message Delays Varying More Than 3 msec. Proc.of *WRTP'99, 24th IFAC/IFIP Workshop on REAL TIME PROGRAMMING.*

Wedde, H.F., J.A. Lind and G. Segbert (1999). Distributed Real-time Task Monitoring In The Safety-Critical System MELODY. Proceedings of the EUROMICRO '99 Workshop on Real-Time Systems, to be published.