INCENSE: Information-Centric Run-Time Support for Component-Based Embedded Real-Time Systems [*]

Andreas Hjertström, Dag Nyström, Mikael Åkerholm, and Mikael Nolin
Mälardalen Real-Time Research Centre, Västerås, Sweden
{andreas.hjertstrom, dag.nystrom, mikael.akerholm, mikael.nolin}@mdh.se

Abstract

In this paper we present a technique to allow the use of real-time database management together with component-based software development, to achieve an information centric run-time platform for the development of embedded real-time systems. The technique allows components to benefit from the advantages of a real-time database management system while still retaining desirable component properties, such as isolation and a high level of reusability. We propose that a database is integrated in the component framework, and introduce the concept of database proxies to decouple components from the database schema. The resulting system fully benefits from the advantages of component-based software development, such as reusability, all component interaction through interfaces, etc, combined with the advantages of a real-time database management system, i.e., system openness, controlled data access, and dynamic query language capabilities.

1. Introduction

Today's vehicle systems have an increasing number of computer nodes, called Electronic Control Units (ECUs), often developed by different hardware vendors, controlling engine, brakes, gearbox etc. The cost for development of electronics in high-end vehicles have increased to more than 23% of the total manufacturing cost [4]. Including subsystems a modern automotive system can contain over 70 ECUs communicating on different networks and exchanging up to 2500 signals [6]. The continuous increase of ECUs and exchanging of signals, leads to an growing amount of data that needs to be managed. A significant amount of the tasks using this data are critical hard real-time transactions, often operating at high frequencies and updated periodically. Furthermore, current trends also show an increase of tasks running non-critical, soft transactions in the system at lower frequency. These transactions, often read transactions, use the same data as hard critical tasks for logging or to present statistical information about the current state of the vehicle to the user.

To handle the increasing complexity in these systems, new approaches and design paradigms to reduce complexity are needed, since current techniques (internal data structures) are becoming increasingly insufficient. Two upcoming approaches to reduce complexity are Component-Based Software Engineering (CBSE) and DataBase Management Systems (DBMS). Real-Time Database (RTDB) [10] and RTDBMS (Real-Time Database Management System) are upcoming technologies both within research society and in industry [5] to help developers solve information management problems regarding synchronization, deadlock and persistency. This area has mainly been focused towards concurrency-control, temporal consistency, overload management and scheduling. The focus within CBSE is to create software components that are reusable entities mounted together as building blocks with a possibility to maintain and improve systems by replacing individual components [3]. Even though RTDBMS and CBSE share the same goal, their means of achieving it is unfortunately conflicting. One key philosophy for most component models is that all communication with the surrounding environment should be performed through the component's interface, eliminating all component side-effects. By introducing a real-time database management system (RTDBMS) and giving components direct access to shared data in the database introduces such side-effects. Furthermore, constructing components that query a certain database engine, with a certain database structure (schema) severely reduces the possibility of component reuse. To remove conflicts between RTDBMS and CBSE we introduce INCENSE (information-centric run-time support for component-based embedded real-time systems), a framework which combines the best of these technologies.

It would be desirable to achieve a component-based system where data is reliably managed and structured to enable

[*]This work is supported by the Swedish Foundation for Strategic Research within the PROGRESS Centre for Predictable Embedded Software Systems.

```
 TASK oilTemp(void){
    //Initialization part
    int temp;
1   MimerDbP dbp;
2   MimRTDBPBind(&dbp,"Select TEMP from
                      ENGINE where
                      SUBSYSTEM='oil'");
    //Control part
    while(1){
3      temp=readOilTempSensor();
4      MimRTWriteInt(dbp,temp);
       waitForNextPeriod();
    }
 }
```

Figure 1. An I/O task that uses a Mimer RT database pointer.

flexibility, a system where soft and hard real-time tasks can execute and keep isolation properties, a system that can handle critical transactions and at the same time enable openness, a system where new functionality can be added or removed without side effects to the system. To achieve this we propose to use a RTDBMS, in this case Mimer SQL Real-Time Edition [5], a commercially available[1] real-time database, together with a component technology, in this case SaveCCT, to achieve an information-centric run-time platform.

1.1. Mimer SQL Real-Time Edition

The Mimer SQL Real-Time Edition (Mimer RT) [5] is a real-time database management system intended for applications with a mix of hard and soft real-time requirements. The hard real-time algorithms are based upon the work performed within the COMET research project [8]. Mimer RT uses the concept of database pointers [9] to access individual data elements in an efficient and deterministic manner. For soft real-time database management, standard SQL [2] queries are used. To achieve database consistency without jeopardizing the real-time requirements the 2V-DBP concurrency control algorithm [7] is used. 2V-DBP allows hard and soft transactions to share data independent of each other.

In Figure 1 an I/O task that reads a sensor and propagates it into the database is shown. The task consists of two parts, an initializing part, and a control part. In the initialization part, the database pointer is created (line 1) and associated with a data element (line 2). The `MimRTDBPBind` function executes the query and a direct link to the data element is established. In the control part, the sensor is scanned (line 3) and its value is written to the database (line 4). The `MimRTWriteInt` call uses the direct link and performs the write in constant time.

[1]Mimer SQL Real-Time edition will be available during Q2 2007.

1.2. SaveCCT Real-Time Component Technology

The SaveComp Component Technology (SaveCCT) [1] is described by distinguishing manual design, automated activities, and execution. The entry point for a developer is the Integrated Development Environment (IDE), a tool supporting graphical composition of components, where the application is created. Developers can utilize a number of available analysis tools with automated connectivity to the design tool. SaveCCT is based on a textual XML syntax which allows components and applications to be specified. Automated synthesis activities generate code used to glue components together and allocate them to tasks. Resource usage and timing are resolved statically during the synthesis instead of using costly run-time algorithms. SaveCCT is, as Mimer RT, intended for applications with both hard and soft real-time requirements.

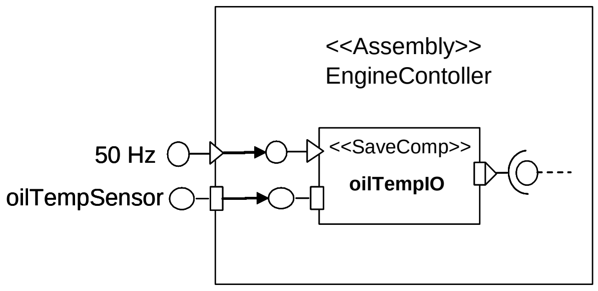

Figure 2. Save graphical application design

In SaveCCT applications are built by connecting components input and output ports using well defined interfaces, see Figure 2. Components are then executed using trigger based strict "read-execute-write" semantics. A component is always inactive until triggered. Once triggered it starts to execute by reading data on input ports to perform its computations. Data is then written to its output ports and outgoing triggering ports are activated.

Figure 2 shows how the XML code in Figure 3 is graphically represented. There are two inports into the Engine Controller application, data and trigger port. Data is read by the oilTempIO component from its inport oilTempSensor once triggered every 50Hz. Computations are done and results propagated onto the output port. In this case the output port is a combined trigger and output port. The XML code also includes ATTRIBUTE which describes the different component properties. In this example we have chosen to exclude all attributes except Worst Case Execution Time (WCET), which is analyzed and entered to the system.

```
<APPLICATION id="EngineController">
   <IODEF>
      <INPORT mode="trig" type="void"
              id="trigFiftyHz" value="20"/>
      <INPORT mode="data" type="int"
              id="oilTempSensor" />
   </IODEF>
   <TYPEDEFS>
      <COMPONENTDESC id="oilTempIO">
         <INPORT mode="trig" type="void"
                 id="trigOilTemp" />
         <INPORT mode="data" type="int"
                  id="oilTemp" />
         <OUTPORT mode="combined" type="int"
                  id="newOilTemp" />
         <ATTRIBUTE id="WCET" type="ms"
                    value="5" />
      </COMPONENTDESC>
      ...
   </TYPEDEFS>
   <CONNECTIONLIST>
      <CONNECTION>
         <FROM id="EngineController"
               port="trigFiftyHz" />
         <TO id="oilTempIO"
            port="trigOilTemp" />
      </CONNECTION>
      <CONNECTION>
         <FROM id="EngineController"
               port="oilTempSensor" />
         <TO id="OilTempIO" port="oilTemp" />
      </CONNECTION>
      ...
   </CONNECTIONLIST>
</APPLICATION>
```

(Figure simplified for readability)

Figure 3. SaveCCT XML description file

2. The Information-centric Component Framework

To efficiently integrate real-time database management and component-based software engineering in order to gain the potential benefits of both approaches, we propose that the RTDBMS is made part of the component framework. Figure 4 shows the architecture of the framework in which it acts as a proxy between the application components and the RTDBMS. This allows components to be *database unaware*. We define a database unaware component as a component which does not have knowledge of the database schema, i.e., the structure of the data in the database. This database decoupling is made possible due to *database proxies*, see Figure 4, which creates a database view that is consistent with the interface of the component.

If components are database aware, i.e., calls to a database is made from within the component-code, a number of unwanted properties emerge, such as (i) decreased component reusability, since the component can only be used in systems with a certain database schema, and (ii) undesirable component side-effects since interaction with the environment is made from outside the component interface.

In this paper, we distinguish between two types of database proxies, namely *hard real-time database proxies* (hard proxies) and *soft real-time database proxies* (soft proxies).

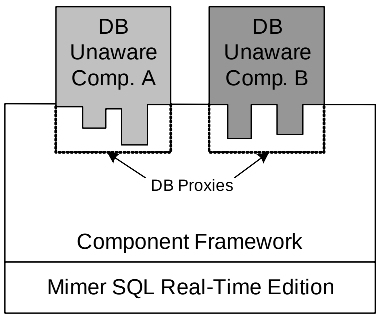

Figure 4. The Incense component framework

2.1. Hard real-time database proxies

Hard proxies are intended for hard real-time components, which need efficient and deterministic access to individual data elements. In Figure 5, a hard proxy is declared as a `DBHARDPROXY`. The declaration contains all information to set up a database pointer, which will be constructed in the component framework as glue code between component calls. Since 2V-DBP provides constant response-time for database pointers, an attribute for worst-case execution time is included in the declaration.

Hard proxies are connected to a component's in- or outport, and the data element in the database is either provided to the component or written back to the database after completion of the component's execution. As a hard proxy can provide a data element of any type, they can be used with any existing components since the database is fully transparent to the component.

2.2. Soft real-time database proxies

Soft proxies are intended for soft real-time components, which might need more complex data-structures. Consider a component monitoring the overall status of a subsystem, e.g., all the temperatures in an engine, or logging of errors etc.

In order for a component to be able to use a soft proxy, it must have a *relational interface*, which means that it must be able to take a relational table as a parameter (or return value). Therefore, the SaveCCT component-model is extended to include `TABLEDESC`'s as parameters, see Fig-

```
<TYPEDEFS>
   <TABLEDESC id="temperatureTable">
      <ATTRIBUTES>
         <ATTRIBUTE type="CHAR(20)"
                    id="subsystem"
                    key="primary"  />
         <ATTRIBUTE type="int"
                    id="temperature"
                    key="false"    />
     </ATTRIBUTES>
   </TABLEDESC>
</TYPEDEFS>

<DBPROXIES>
   <DBHARDPROXY type="int" id="oilTemperature"
                bind="SELECT temp FROM engine
                      WHERE subsystem='oil'"
                <ATTRIBUTE id="WCET" type="us"
                           value="5" />
   />
   <DBSOFTPROXY type="temperatureTable"
                id="engineTemperatures"
                bind="SELECT temp from engine"
                <ATTRIBUTE id="WCET" type="ms"
                           value="3" />
   />
</DBPROXIES>
```

Figure 5. SaveCCT proxy representations

ure 5. A TABLEDESC table descriptor is a relational table containing the information needed by the component. It is worth noting that the structure of this descriptor is completely decoupled from the database schema.

Soft proxies are, as hard proxies, connected to a component's in- or out-ports. At run-time, the soft proxy converts the database schema into the format of the table descriptor. This glue-code, i.e., the database query associated with the proxy, is embedded into the component framework.

This approach implies that the component is aware that an RTDBMS is present, but it is still generic with respect to the schema of the database, i.e., component reusability is maintained.

3. Conclusions and Future Work

In this paper we present a technique to integrate a real-time database management system into a component-based system, and thereby gaining the advantages of high level data management while retaining important properties, such as component isolation and reusability, of component-based software engineering. We introduce the concept of a *database proxy* to enable components to be database unaware, i.e., components do not need to be tailored to fit a certain database engine or database schema. Instead all database interactions are performed from the information-centric component framework used in this technique.

Key benefits of this approach include, system openness due to standardized query language languages, the possibility of creating on-the-fly dynamic database queries, total decoupling of data and components. Furthermore, when using a hard real-time database engine such as Mimer SQL Real-Time edition, hard real-time guarantees on data access is provided and synchronization of shared data is transparently managed. By separating data access and components, component isolation is retained and managed by application specific database proxies connected to component interfaces.

In our future work we intend to extend the information-centric view with high level tools and design paradigms to manage and organize data in a logical view rather than a physical. During design, developers should have full control of each data item involved, who are the producers/consumers, timing requirements etc. The overall aim of our work is to create an information-centric design paradigm for real-time systems, where data management is treated as its own design entity.

References

[1] M. Åkerholm, J. Carlson, J. Fredriksson, H. Hansson, J. Håkansson, A. Möller, P. Pettersson, and M. Tivoli. The Save Approach to Component-Based Development of Vehicular Systems. *Journal of Systems and Software*, 2006.

[2] S. Cannan and G. Otten. *SQL - The Standard Handbook*. MacGraw-Hill International, 1993.

[3] I. Crnkovic. Component-Based Software Engineering - New Challenges in Software Development. *Software Focus*, December 2001.

[4] L. Gabriel and H. Donal. Expanding Automotive Electronic Systems. *Computer*, 35(1):88–93, Jan 2002.

[5] Mimer SQL Real-Time Edition, Mimer Information Technology. Uppsala, Sweden. http://www.mimer.se.

[6] N. Navet. Trends in Automotive Communication Systems. In *Proceedings of the IEEE*.

[7] D. Nyström, M. Nolin, A. Tešanović, C. Norström, and J. Hansson. Pessimistic Concurrency Control and Versioning to Support Database Pointers in Real-Time Databases. In *Proceedings of the 16th Euromicro Conference on Real-Time Systems*, 2004.

[8] D. Nyström, A. Tešanović, M. Nolin, C. Norström, and J. Hansson. COMET: A Component-Based Real-Time Database for Automotive Systems. In *Proceedings of the Workshop on Software Engineering for Automotive Systems*, pages 1–8. The IEE, June 2004.

[9] D. Nyström, A. Tešanović, C. Norström, and J. Hansson. Database Pointers: a Predictable Way of Manipulating Hot Data in Hard Real-Time Systems. In *Proceedings of the 9th International Conference on Real-Time and Embedded Computing Systems and Applications*, pages 623–634, February 2003.

[10] K. Ramamritham, S. H. Son, and L. C. Dipippo. Real-Time Databases and Data Services. *Journal of Real-Time Systems*, 28(2/3):179–215, November/December 2004.