PARTITIONING METRICS FOR IMPROVED PERFORMANCE AND ECONOMY OF DISTRIBUTED EMBEDDED SYSTEMS

Augustin Kebemou
Fraunhofer-Institut für Software- und Systemtechnik (ISST)
Mollstrasse 1, 10178 Berlin, Germany
augustin.kebemou@isst.fraunhofer.de

Abstract: *This paper describes some metrics which can be applied to optimize the constrained partitioning of very large distributed software/hardware systems. These metrics are tailored for software component models for UML specifications. The use of clear defined metrics allows us to capture more aspects of the design (abstraction level, aggregation and causality relations between the components). Besides, the appropriate formalization of these metrics has a great impact on the results that can be obtained from the partitioning regarding both the performance and the economy of the system under design.*

Keywords: Software Component, Metrics, Codesign, Partitioning, Automotive

1. INTRODUCTION

Embedded systems are used in live facilities where the principal requirements include safety, performance and security. A very large embedded system may consist of several parts which are geographically separated, like those in a car. Automotive systems are complex systems with software and hardware that are constantly in execution. The functionalities of such systems are located on different physical components (subsystems) which do not only cooperate with each other, but also depend on each other. Anyhow, it will be necessary to design all the parts of the system simultaneously to ensure that they will conjointly meet the given performance and economy goals. This is otherwise called codesign. Thus, effective design methods are required.

Each of the functionalities expected from a system can be specified by a function which implements it. Hardware/software codesign proposes a concurrent design of the hardware and the software parts of the system in order to meet the best trade-off between performance and costs. Some really large embedded systems offer a very big set of functionalities which can only be modeled by thousands of high level functions. For these systems, the modeling languages commonly used in the current codesign methodology for the partitioning offer a poor structuring, since the abstraction level is comparable with the one of programming languages like C or Ada [9]. We overcome this scarceness by using a modeling language which can handle the specification of the artifacts of very large systems at every abstraction level. We use the unified modeling language UML, which in the new version 2.0 extended with the component-based design paradigm simplifies the modeling of software/hardware systems [4] through a port concept. The port concept provides abstraction by means of aggregation mechanisms and information hiding principles [2, 8] for the design complexity management. Furthermore, the UML provides a graphical visualization of the logical system architecture independently of the granularity of the specification.

The partitioning of the specification of a large technical system is a challenging task since it is not enough to partition the functionalities of a system among the various subsystems; more important is to achieve a partition which satisfies the system constraints. Thus the challenge of the partitioning task is to find the best system architecture including the right distribution of the functions among the subsystems and the right hardware components and communication facilities.

We propose a partitioning heuristic the input of which is a functional logical specification. We use a set of metrics to guide the partitioning of the specification before we conjointly design the subsystems which we have methodically determined. This paper will particularly present the partitioning metrics we use at the early steps of the partitioning task.

The paper is organized as follows: The next section presents the motivations of the metric guided partitioning in the automotive engineering and a summary of some works considering the necessity of measurements in the system development. The section 3 introduces briefly a component-based system specification metamodel defined for mixed software/hardware Systems. The following section presents the partitioning metrics and section 5 presents some concrete advantages derived from a metric guided partitioning application in the automotive engineering.

2. RELATED WORK AND MOTIVATION

A typical example of the complexity of embedded systems is the amount of Electric/Electronic (E/E) systems providing the comfort, the performance and the security in the automobile. It is quite difficult to imagine what the car of the future will look like. Even if ESP (Electronic Stability Program), ACC (Active Cruise Control), ABS (Anti Blocking System for the brakes) and airbags are nowadays mainstream, it is clear that on the way to build the cars which will drive autonomously, more and more functionalities will be added to the system automobile. The mastering of the design of such systems is a great challenge because the system consists of subsystems which are distributed on different physical locations, needing to cooperate with each other in order to fulfill the required functionalities. Some important design constraints can be formulated as follows: the monetary cost of the product with regard to the time to market and the material usage have to be hold minimal by providing concurrently high functionality and performance. The volume, the weight and the energy consumption of automotive systems must not grow proportionally to the number of the embedded functionalities.

In the current practice, the OEM (Original Equipment Manufacturer) partitions the functionalities according to vague estimations or to experience, without any helpful documentation. Then he orders some electronic control units (ECU), each implementing a part of the functionalities and plug them together. Consequently, the real time constraints are not easy to achieve, due to the communication difficulties between sensors, ECUs and actuators. There are more material resources installed into the system as obligatory necessary making the cost/performance ratio poor. To solve the communication problems, they design better and better communication protocols (e.g. CAN and family, MOST, Flex Ray, LIN) or they add the communication bandwidth by adding new buses. This alone will not definitely solve this problem actually challenging the industrial research. Suitably localized functions communicate cheaper. A goal-oriented approach of the partitioning, which can be automated is the key to master the costs and the performance requirements of automotive systems.

Since system level models are abstractions of implementation, metrics of interest are required to evaluate system models and the partitioning state space of such mix software/hardware systems. Metrics are commonly used to estimate/predict the costs and schedules of the product and the process by which it is developed. Product metrics measure the product quality at any stage of the development. In the software engineering, product metrics may measure the size and the complexity of either the model or the final code [3, 7, 18]. Process metrics may estimate the overall development time, the average experience of the staff or the type of the methodology used. Software process metrics are summarized in [5, 10]. Particularly, some general metrics have been proposed [6, 10, 11, 15] to evaluate the quality of object oriented system design. But the known software and OO design metrics concentrate on the characterization of software programs or OO Models. They can not consider the constraints of heterogeneous software/hardware systems.

Two key metrics are used in the software/hardware codesign partitioning to define the quality of a partition: performance and hardware size. Since the partitioning in the software/hardware codesign is basically the process of deciding, for each subsystem, whether the required functionality is more advantageously implemented in hardware or in software, these metrics mostly address especially the 2-way clustering issues. Closeness metrics are used in [12, 14] to predict the benefit of grouping any two specification objects. Closeness metrics have been defined on fine-granularity : logical, arithmetical operations or assignments [16] and on coarse-grainularity obejcts [13]. In a great work, Vahid and Gajski [17] presented some "closeness metrics for the system-level functional partitioning", with VHDL-level specifications input. We redefined these metrics to close the area of distributed systems with geographical separation constraints. Our metrics are tailored for the particular concept of software components in order to enable the investigation (structural analysis) of UML-level models for constrained distributed embedded systems.

3. INPUT PRESENTATION

We specify large systems using a ULM-like modeling language. At the logical level, a set of system functionalities are represented by logical functions. We encapsulate the logical functions in software components. A software component has an internal behavior, an input and an output interface. The component interfaces are the communication facilities of the components. They are represented by two sorts of ports: the input ports at the input interfaces and the output ports at the output interfaces. The inter-component communication is modeled by means of signal transfer over logical connectors, which we simply name connectors. Each connector represents the fact that in a logical channel the connected components share a set of signals called connector interface. With this architecture, the communication can be modeled independently from the internal behavior of the components. This components concept can be used at every granularity level of the specification. So we can build a logical model of the system under construction as a network of communicating components. Figure1 shows the metamodel of the logical component network and Figure2 shows a part of a functional specification.

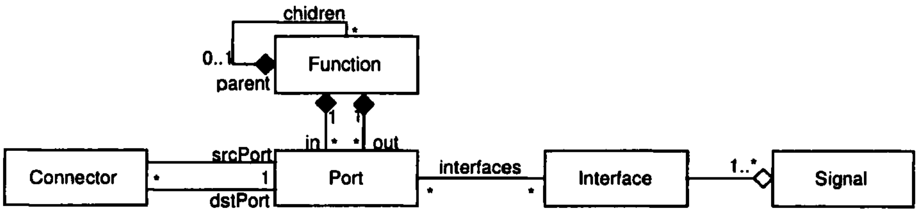

Figure 1. The metamodel of the logical specification

We add descriptive attributes to our model elements. The attributes describe the implementation requirements and the external behavior of the components, especially their communication requirements. The required memory, the maximum execution time, the average execution period of the function illustrate the kind of information which describe a function. Ports are attributed by the maximal cycle time of signal calls, the security level, etc. Each signal has a resolution, a priority, a transmission/access frequency (accfreq) and so on. The logical model gives sufficient information at least to measure the closeness between the functions. That is to predict the benefits of grouping any two logical components. The partitioning can now be driven.

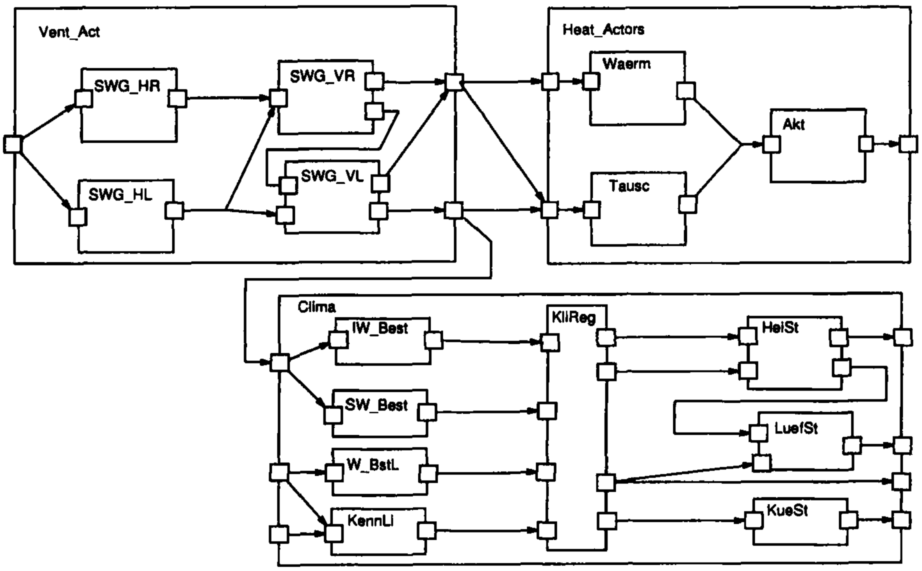

Figure 2. Example of a functional specification

We can formally define the functional model as a tuple as follows:

$G = \langle F, P, S, C \rangle$
$F = \{components\}$
$P = I \oplus O$; $I = \{input\ ports\}$ and $O = \{output\ ports\}$
$S = \{signals\}$; $signal = \langle src, dst \rangle$ where $src, dst \in F$
$C = \{connectors\}$; $connector = \langle src, dst, Int \rangle$
where $src \in O$, $dst \in I$ and $Int \subseteq S$.

This is a graph which nodes represent the functions/components of the specification and the edges are the connectors. The ports are inherent parts of the nodes. Each connector is labeled with the signals it transports.

4. PARTITIONING METRICS

Since the specification language provides a component de-/composition mechanism (hierarchical aggregation), we define the partitioning metrics between two sets of components. A set of components can be seen as a bundle of components of the same abstraction level or a package. For illustration, figure 2 shows 3 sets of components. We use appropriate normalization factors in order to combine and to fairly compare the values resulting from the measurements.

4.1 The connection metric

The connection measures the number of interfaces shared between two sets of logical components. It is advantageous to implement two functions which have a common great number of communicating interfaces together on the same component. Doing so, we intent to reduce the inter-components communication. The connection of two sets of components is the number of connectors connecting a function of the first set with a function of the second set.

Assume that the operator $AccessedConnectors(f)$ returns the set of connectors transporting at least one signal s for which $s.src \in f$ or $s.dst \in f$;
$AccessedConnectors(f) = \{c \in C | \exists s \in c.Int; s.src \in f \vee s.dst \in f\}$
The connection metric is formalised as follows:

$$\begin{aligned}
Connection(F_i, F_j) &= |commonConnectors_{i,j}|/Norm \\
commonConnectors_{i,j} &= C_i \cap C_j \\
C_i &= \textstyle\bigcup_{f \in F_i} AccessedConnectors(f) \\
C_j &= \textstyle\bigcup_{f \in F_j} AccessedConnectors(f) \\
Norm &= |C_i \cup C_j|(local) \\
&= |C|(global)
\end{aligned}$$

C_i respectively C_j represent the sets of the logical connectors which are associated with the ports of the functions F_i respectively F_j.
$commonConnectors_{i,j}$ is the set of connectors commonly accessed by F_i and F_j. The local normalization factor is the set of the connectors accessed by F_i or F_j. $Connection(F_i, F_j)$ is the normalized number of connectors accessed by both F_i and F_j.

4.2 The communication metric

The communication measures the estimated amount of data transferred between two sets of components. The communication provides a different information as the connection. For example, if two components communicate 8 bytes of user data over a connector and 2 bytes over another one, the connection will consider two connectors, whereas the communication will consider the total number of bits shared between the two components (10 bytes). It is important to keep this metric different from the connection because the connectors are often differently constrained. We compute the communication by the means of the operator $AccessedConnectors$.

$$\begin{aligned}
Communication(F_i, F_j) &= com_{i,j}/Norm \\
com_{i,j} &= \textstyle\sum_{c \in commonConnectors_{i,j}} comWeight(c) \\
comWeight(c) &= \textstyle\sum_{s \in c.Int} s.accfreq \times s.resolution \\
Norm &= \textstyle\sum_{c \in C_i \cup C_j} comWeight(c)(local) \\
&= \textstyle\sum_{c \in C} comWeight(c)(global)
\end{aligned}$$

$comWeight(c)$ represents the amount of data (in bits) transfered through the connector c during an average activation time of the component $c.src$. $com_{i,j}$ is the total amount of data shared between the function sets F_i and F_j during an average activation time of the functions F_i and F_j. The local normalization factor is the total amount of data transferred between F_i, F_j and any other function of the specification whereas the global normalization factor is the total amount of data transferred around all the specification during an average activation time of the functions F_i and F_j.

4.3 The common accessors metric

Grouping two sets of components which have a great common set of accessors (e. g. a sensor) can improve the quality of the communication and therefore the performance of the system.

Let's assume that the operator $Accessors(f)$ returns the set of logical components $(f_i) = \{f_1, f_2, \dots f_n\}$ for which exists at least one signal s so that $(s.src = f$ and $s.dst \in (f_i))$ or $(s.src \in (f_i)$ and $s.dst = f)$. We observe that each function is its own accessor (i. e. $\forall f; f \in Accessors(f)$).

$$\begin{aligned}
CommonAccessors(F_i, F_j) &= |commonAccessorSet_{i,j}|/Norm \\
commonAccessorSet_{i,j} &= allAccessor_i \cap allAccessor_j \\
allAccessor_i &= \textstyle\bigcup_{f \in F_i} Accessors(f) \\
allAccessor_j &= \textstyle\bigcup_{f \in F_j} Accessors(f) \\
Norm &= |allAccessor_i \cup allAccessor_j| \text{ (local)} \\
&= |F| \text{ (global)}
\end{aligned}$$

$allAccessor_i$ represents the set of logical components that share at least one signal with a function contained in F_i. $commonAccessorSet_{i,j}$ is the set of all components communicating with both a function contained in the component F_i and a function contained in F_j. $CommonAccessors(F_i, F_j)$ returns the normalized number of the components accessing F_i and F_j, where the local normalization factor is the number of the logical functions communicating with a function contained in F_i or in F_j and the global normalization factor is the number of all components of the specification.

4.4 The communication constraint metric

Embedded systems typically are subject to real time, hard security and performance constraints which are considerably influenced by the inter-components communication. The precedent metrics have not taken these important constraints into consideration. When partitioning the logical specification of highly constrained heterogeneous systems, the number of channels and the amount of data shared between two components shall not always be sufficient as closeness indicators. For example, although the brake control unit communicates more often with other functions than with the crash-avoidance sensor, the designer would decide to favorite this relatively rarely activated connector on the basis of the interpretation of the values of the communication constraint metric.

We define $cons$ as the set $\{e_1, e_2, \ldots, e_n\}$ of the given communication constraints. Since we globally map the connectors rather than the signals on buses, we assume that all signals transferred over a connector are equally constrained. That's what we mean bellow by constrained connectors ($consConnectors$). We assume that the operator $consConnectors_{e,i,j}$ returns the set of the connectors common to the component sets F_i and F_j that are subject to a given constraint e.

$$\begin{aligned}
ComCons(F_i, F_j) &= \textstyle\sum_{e \in cons} \alpha_e Cons_e(F_i, F_j) / \sum_{e \in cons} \alpha_e \\
Cons_e(F_i, F_j) &= consCom_{e,i,j} / Norm \\
consCom_{e,i,j} &= \textstyle\sum_{s \in commonConsSig_{e,i,j}} s.accfreq \times s.resolution \\
commonConsSig_{e,i,j} &= \textstyle\bigcup_{c \in consConnectors_{e,i,j}} c.Int \\
Norm &= \textstyle\sum_{e \in cons} consCom_{e,i,j} \text{ (local)} \\
&= \textstyle\sum_{e \in cons; F_i, F_j \in F} consCom_{e,i,j} \text{ (global)}
\end{aligned}$$

$commonConsSig_{e,i,j}$ (common constrained signals) is the set of signals exchanged between the component sets F_i and F_j which are involved in the achievement of the constraint e. $consCom_{e,i,j}$ (constrained communication) is the volume of data (in bits) exchanged between the component sets F_i and F_j over the connectors which are subject to the constraint e. $Cons_e(F_i, F_j)$ is the normalized heaviness of the communication between F_i and F_j which is constrained by e. The local normalization factor is the volume of data shared between the component sets F_i and F_j over all common connectors, each being subject to at least one of the given communication constraints. The global normalization factor is the volume of the constrained data transported around the whole specification. $ComCons(F_i, F_j)$ considers all chosen constraints on the communication between the component sets F_i and F_j and returns the normalized communication constraint.

The designer is free to choose the values parameter α_e that reflect the weight and the importance of each constraint e for a given communication.

4.5 The sequenceness metric

Implementing two components which are defined to run sequentially on the same processor is obviously advantageous. Thus, grouping sequentially executing components of the logical functional specification will take benefits from these advantages.

Let's assume that the operator $Sequent(f_1, f_2)$ returns 0 if f_1 and f_2 could be executed concurrently and 1 if not.

$$\begin{aligned}
Sequenceness(F_i, F_j) &= sequentPairs_{i,j}/Norm \\
sequentPairs_{i,j} &= \textstyle\sum_{f_1 \in F_i, f_2 \in F_j} Sequent(f_1, f_2) \\
Norm &= |F_i| \times |F_j| \text{ (local)} \\
&= \frac{|F| \times (|F|-1)}{2} \text{ (global)}
\end{aligned}$$

$sequentPairs_{i,j}$ is the number of pairs of functions contained in F_i x F_j which could execute sequentially. The local normalization factor is the number of all possible pairs of functions of the set F_i x F_j. The global normalization factor represents the number of all pairs of functions around the whole specification.

4.6 The resource sharing metric

The resource sharing metric measures the likelihood that two logical components should share the same resource. A resource can be a physical hardware or any hardware-closed logical object like message frames (shared between signals) in the case of serial bus transmission. If the resource sharing constraint is not dictated from the system requirements, i. e. some functions are designed a priori to run on the same physical component while other pairs of functions must exclude each other (for example because of electromagnetic incompatibilities), we can hardly measure the resource sharing with the available information. An example of resource sharing criteria could be the fact that different functions which never run at exactly the same time could use the same hardware resource (e. g. a reconfigurable processor, FPGA). Just as well, if some components are supposed to implement each a particularly constrained algorithm for which they need an adequate optimized processing element to run, it will be likely to group them together. This is the case of using an ASIP for the optimized computation paths or ASSPs, ASICs. Therefore, although the partitioning is done on the logical structural specification, the resource sharing metric can not only be evaluated from the structure of the system. It also depends on the components behavior. Details will be presented in a coming work.

5. RESULTS

We applied the metrics on a functional specification containing 4 functions representing the features of a part of the chassis of a car. The specification was originally partitioned and mapped on four different ECUs communicating through a CAN Bus. CAN [1] is an event triggered bus concept used in the automotive industry, by which the messages are identified rather than the sender. After we decomposed the aggregations contained in the model, we clustered the resulting refined specification of 18 low level atomic functions in a bottom-up manner using a simple closeness function adding different metrics unweighted together. The partitioning function only considered the connection, the communication and the communication constraint metric.

To investigate the effects of the partitioning on the communication, we had to consider both the performance of the system and the order of the messages necessary for the inter-components communication that results from the dependencies between the tasks loaded on different ECUs. The performance has been measured by the ability of the bus to react to asynchronous (i. e. which are not predictable) external events at the working time. That is the possibility for the controller to access the bus within a predefined period of time after buffering the signal corresponding to an asynchronous event requiring emergency handling, in order to write a "CAN-message" containing this signal. Short latencies are easy to achieve in such a low bus load constellation, when the message priorities are consequently distributed. Since we were interested in the dependability of the system upon the economy of data transfer more than on other performance attributes, the most significant result was the number of messages transmitted over the bus.

For the measurements, we fixed the data rate on the bus to 250 kbits/s and the message length in the range from 0 to 4 bytes user data. Each of the different system architectures have been proved to realize the principal functionalities of the system. These include the mean computation time for the constrained functions, the maximum response time of the components (i.e latency due to the time to wait for the permission to access the bus), the transmission delay, which depends on the data rate, the length of the message and the topology of the system. All of them important as automotive systems underly real-time requirements [19]. We exited the functions in a way that we could simulate the maximum bus work case corresponding to each partition, by stimulating the functions in a way that every component would transmit the maximum number of messages on the bus as it was possible, in order to simulate the worst case of the bus regular operation regarding its load. The external/environmental events have been created by artificial functions representing some virtual sensors. Under these conditions all asynchronous events could be handled. This was not surprising as we mentioned above, simply because of the reduced size of the specification under experimentation, but could not hide the significance of the results of our interest, namely the number of messages exchanged through the bus. For the partitions of 1,..,6 parts, table 1 shows the approximated augmentation(+)/ diminution(-) in percent of the number of messages exchanged through the bus in comparison to the original 4-parts partition.

Number of parts	1	2	3	4	5	6
Variation of the inter-components communication	-100	-12	-5	0	+21	+45

Table 1. Effects of the partitioning on the serial bus communication

The partition with 2 parts appeared to be the best one when combining the inter-components communication and the geometrical advantages (i. e. peer-to-peer communication with sensors, gateways and actuators which are not in the bus network).

6. CONCLUSION AND COMING WORK

We have defined some powerful metrics which we use to distribute a software component-based specification on different physical components. These metrics are particularly useful when the different components must be geographically separated since they optimize the inter-components communication constraints. The metrics aim at helping the experienced designer to verify and document his assumptions, while the novice will get guidance in the partitioning task. Since the resource sharing requirements depend at least partially on the functional behavior of the components and on the mapping and scheduling strategies, a next paper will introduce further useful metrics additionally to the behavioral specification of such very large, highly constrained distributed systems and thereupon will complete the opened metrics definitions.

REFERENCES

[1] *CAN Specification Version 2.0, Part A, Part B; Robert Bosch GmbH.*

[2] *Systems Modelling Language (SysML) Specification, OMG UML for SE RFP, 2004.*

[3] A. J. Albrecht and Jr. J. E. Gaffney. Software Function, Source Line of Code, and Development Effort Prediction: A Software Science Validation. *IEEE Trans. Software Eng., Vol 9 Number 6, pp. 639-648,* 1983.

[4] M. Bjoerkander and C. Kobryn. UML 2.0 - Der nächste Schritt. *Elektronik Automotiv, Germany; pp. 30-33,* April 2002.

[5] B. W. Boehm. Software Engeeniering Economics. *IEEE Trans. Soft. Eng. Vol 10, Number 1,* 1984.

[6] S. R. Chidamber, D. P. Darcy, and K. F. Kemerer. Managerial Use of Metrics for Object-Oriented Software: An Exploratory Analysis. *IEEE Transactions on Soft. Eng. Vol 24, Number 8, pp 629-639,* 1998.

[7] M. H. Halstead. *Elements of Software Science.* New York: Elsevier North-Holland, 1977.

[8] M. Jeckle, C. Rupp, J. Hahn, B. Zenger, and S. Queins. *UML 2 Glasklar.* Hanser Verlag ISBN 3-446-22575-7, 2004.

[9] R. Kamdem. *Contribution au Partitionement Materiel/Logiciel des Systemes Temps Reel pour les Applications de Controle d'Experiences.* PhD thesis, Univ de provence; France, 1999.

[10] Chris. F. Kemerer. An Empirical Validation of Software Cost Estmation Models. *ACM, Vol 30, Number 5 , pp. 416-429,* 1987.

[11] M. Marchesi. OOA Metrics for the UML. *Proc. of the 2nd Euromikro Conference on Software Maintenance and Reengineering,* 1998.

[12] M. C. McFarland. Using bottom-up design techniques in the synthesis of digital hardware from abstract behavioral descriptions, pp 474 - 480. *Annual ACM IEEE Design Automation Conference archive Proceedings of the 23rd ACM/IEEE conference on Design automation,* 1986.

[13] R. Niemann and P. Marwedel. Hardware/Software Partitioning Using Integer Programming. *IEEE/ACM Proc. of EDAC 96, pp 473-479,* 1996.

[14] C. A. Papachristou and W. Zhao. Architechtural Partitioning of Control Memory for Application Specific Programmable Processors. *Proceedings of the 1995 IEEE/ACM international conference on Computer-aided design, December 1995,* December 1995.

[15] R. Reissing. *Bewertung der Qualität objectorientierter Entwürfe.* PhD thesis, Uni Stuttgart; Germany, 2002.

[16] W. Rosenstiel, E. Barros, and X. Xion. A Method for Partitioning UNITY Language in Hardware and Software. *Proc. of The European Conf. on Design Automation,* 1994.

[17] F. Vahid and D.D. Gajski. Closeness metrics for system-level functional partitioning. *IEEE Proceding 1995, 328-333,* 1995.

[18] Tong Yi, Fangjun Wu, and Chengzhi Gan. A Comparison of Metrics for UML Class Diagrams. *ACM SIGSOFT, SE Notes, Vol 29, Number 5,* Sep 2004.

[19] K. M. Zuberi and K. G. Shin. Scheduling Messages on Controller Area Network for Real-Time CIM Applications. *IEEE Trans. Robotics and Automation, pp 310-314,,* April 1997.