ABSTRACT COMMUNICATION MODELING

A Case Study Using the CAN Automotive Bus

Gunar Schirner and Rainer Dömer
Center for Embedded Computer Systems
University of California Irvine
hschirne@uci.edu, doemer@uci.edu

Abstract Communication modeling is a critical issue in specifying SoCs. It is needed for accurately predicting the timing behavior of the system. Fast simulation capabilities are a key in this environment, for coping with the complex design choices during the specification process. Recently, Transaction Level Models (TLM) have been proposed to speedup communication simulation at the cost of accuracy.

This paper reports on a case study, where an automotive communications protocol, the Controller Area Network (CAN), has been captured at different levels of abstraction, where specific features of the protocol, such as bit stuffing, are reflected in the model, or abstracted away. The resulting models have been measured in an experimental setup in terms of performance and accuracy. The paper will analyze the results and evaluate the benefits and drawbacks of these TLM and pin-accurate models. In conclusion it will be shown for which applications the models are suitable, with respect to their speed/accuracy trade off.

Keywords: Transaction Level Modeling, Communication Modeling

1. Introduction

The System-On-Chip (SoC) design faces a gap between the production capabilities and time to market pressures. The design space, to be explored during the SoC design, grows with the improvements in the production capabilities, while at the same time shorter product life cycles force an aggressive reduction of the time-to-market. Addressing this gap has been the aim of recent research work. As one approach, abstract models have been introduced to tackle the design complexity.

Fast simulation capabilities are required for coping with the immense design space that is to be explored; these are especially needed during

early stages of the design. This need has pushed the development of Transaction Level Models (TLM) [6], which are abstract models that execute dramatically faster than synthesizable, bit-accurate models.

Transaction level modeling, however will come with the drawback of a decreased accuracy. This paper will analyze the performance gains of transaction level modeling and show the drawbacks in accuracy. The analysis is based on a case study of the Controller Area Network (CAN) bus, which is a standard bus protocol used in the automotive industry.

This paper will first introduce the main features of the CAN bus. Based on a feature selection a set of models with different levels of abstraction will be proposed and their design will be described. Following that the implemented models will be measured in an experimental setup and their results will be analyzed, to conclude with a set of models suitable for the desired application.

1.1 Related Work

System level modeling has become a more important issue over the recent years, as a means to improve the SoC design process. Languages for capturing these models have been developed, such as SpecC [3] or SystemC [6]. Furthermore capturing and designing communication systems using transaction level models has received research attention.

Sgroi et al. [11] address the SoC communication with an Network-on-Chip (NoC) approach. They propose partitioning of the communication into separate layers that follow the OSI structure. Software reuse is promoted with an increase of abstraction from the underlying communication framework.

Siegmund and Müller [12] describe with SystemC^{SV} an extension to SystemC, and propose modeling of an SoC at different levels of abstraction. They describe three different levels: the physical description at RTL level, a more abstract model that covers individual messages, and a most abstract level that deals with transactions.

[1] describes how the CAN bus is modeled using the above mentioned extension SystemC^{SV}. The work also shows the three abstraction levels, but does not give any experimental results on performance or accuracy.

In [2] Caldari et al. describe the results of capturing the AMBA rev. 2.0 bus standard in SystemC. The bus system has been modeled at two levels of abstraction, fist a bus functional model on RTL level and second a model on TLM level. Their TLM model reached a speedup of 100 over the RTL level model.

2. Introduction CAN Bus

The Controller Area Network (CAN) is a serial communications protocol, introduced by the Robert Bosch GmbH [10], that was designed with a focus on automotive applications.

CAN is a serial multi master broadcast bus. Messages, with up to 8 bytes user data, are received by all bus nodes and distinguished by the message identifier. Each bus node decides using local rules whether to process the message. The message identifier also serves as a message priority. If multiple senders attempt a transmission, the collision free CSMA/CA arbitration will guarantee that the highest priority message will succeed undisturbed.

The CAN bus defines two bus states: recessive (1) and dominant (0). A CAN data frame has the basic format shown in Figure 1. After transmitting the start of frame bit, the message identifier is transmitted with the most significant bit first. During transmission, each sender compares the send and receive signal. A sender that has send recessive bit but a detects a dominant bit will back off from transmission. Another sender must have started a higher priority message.

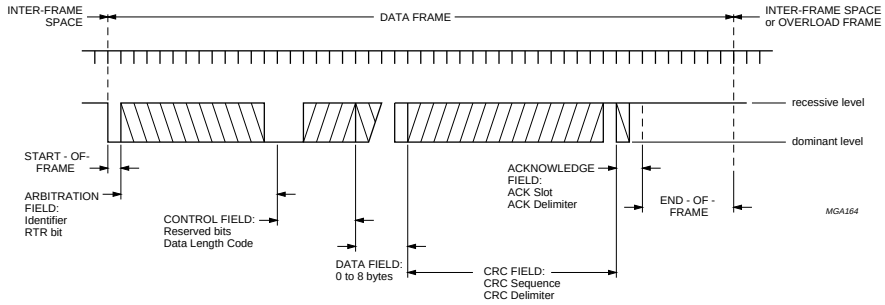

Figure 1. CAN Data Frame (Source [9]) .

In order to ensure correctness of the received data, each CAN message includes a 15-bit CRC. In case of a CRC mismatch, a retransmission of the frame is triggered. The protocol also defines elaborate error detection and error confinement rules for protection against faulty bus nodes.

The CAN serial protocol operates without a centralized clock. Each bus node synchronizes on the bit stream of the sender. A bit stuffing rule guarantees sufficient edges for this synchronization. After transmitting 5 bits of equal polarity, a bit of opposite polarity is introduced.

In summary, the following properties are candidates for abstraction:

- Serial protocol
- Bit synchronization
- Error detection and confinement
- Bit error detection using a 15 Bit CRC
- Bit stuffing
- Arbitration, bus access controlled by CSMA/CA

The following section describes our modeling of the CAN bus. For each model a subset of the above listed features is selected.

3. Modeling

A layered architecture was chosen for the communication system modeling in order to cope with the complexity of communication. Following the ISO OSI reference model [8], the CAN specification falls within the second layer, the data link layer. For modeling of the CAN bus the media access control (MAC) and the protocol sublayer, both sublayers of the data link layer, are considered as well as the physical layer.

The OSI layer definition is based on functional concerns. An alternative view, suitable for describing the models, focuses on the granularity in which user data is handled. The **media access layer** provides services for the transmission of a contiguous block of bytes, called a **user transaction**. This layer divides the arbitrary sized user transaction into smaller bus transactions and transfers them using the protocol layer. The **protocol layer** transfers data as **bus transactions**, which are bus primitives (e.g. a CAN data frame with up to 8 bytes data), and uses the physical layer services. The **physical layer** implements a **bus cycle** access to sample and drive individual bus wires.

Figure 2 shows how the above defined data granularity levels can be analyzed with respect to time. A user transaction is successively split into the smaller elements: bus transaction and finally bus cycles.

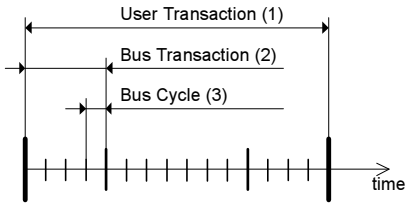

Figure 2. Time decomposition of a user transaction.

Using a system level modeling approach, each layer was implemented as a separate channel using a system description language (SDL)[1].

3.1 Transaction Level Model

The Transaction Level Model TLM is the most abstract model - it only implements the media access layer. The user data, handled at the user transaction granularity, is transferred in one chunk, regardless of the size. The bus access is checked only once per user transaction.

[1]SpecC [3] was used as the SDL of choice, SystemC could be used just as well.

In the implementation, the user data is transferred using a single *memcpy*. The timing is simulated by a single *waitfor* statement, covering the whole user transaction. Neither the CRC nor bit stuffing are observed, since both would require a bit inspection of each message. For an increased performance concurrent bus access is avoided using a semaphore, hence the concurrency resolution relies on the simulation environment and does not observe the message identifier.

3.2 Arbitrated Transaction Level Model

The Arbitrated Transaction Level Model (ATLM) simulates the bus access with a bus transaction granularity (CAN frames), at the protocol layer level. It uses the MAC layer implementation of the later described bus functional model to split user transactions into bus transactions.

The ATLM accurately models the arbitration for each bus transaction (CAN frame) based on the message identifier. It collects all requests during start of frame, and proceeds with the highest priority message. The bus simulation has been implemented without an own flow of execution, in order to maximize execution performance.

Two variants of the ATLM model have been defined. The first, the ATLM (a), performs a bitwise inspection of the frame in order to calculate the CRC and perform stuff bit handling: a stuff bit is inserted/removed each time 5 bits of equal polarity are found. With the bit stuffing, the physical frame length depends on the frame content. The second model, the ATLM (b), does neither calculate the message CRC and nor does it handle stuffing bits. It avoids the costly bit inspection and is expected to execute faster than the ATLM (a), however at the cost of accuracy.

3.3 Bus Functional Model

The bus functional model is a synthesizable model bus model that covers all timing and functional properties of the bus definition. It is a pin accurate and cycle accurate model of the bus.

The bus functional model implements all features of the specification. It protects the data by the CRC, handles stuff bits and performs arbitration. The frame data is send and received serially and the nodes clock is synchronized to the bit stream according to [10] and [7].

Table 1 summarizes the features implemented by a model and shows at which granularity user data is handled. Each model has been implemented in the SDL with the following amount of code lines (excluding testbench): TLM: 250, ATLM (b): 475, ATLM (a): 550, BF: 1400. The model performance and accuracy is analyzed in the following section.

Feature	Bus Functional Model	ATLM (a)	ATLM (b)	TLM
serial transmission	yes	no	no	no
bit synchronization	yes	no	no	no
error detection, confinement	yes	no	no	no
CRC calculation	yes	yes	no	no
bit stuffing	yes	yes	no	no
arbitration	yes	yes	yes	no
data granularity	bus cycle	bus transaction	bus transaction	user transaction

Table 1. Summary of features supported or abstracted away in the models.

4. Analysis

This chapter will explore how the implemented models can be used for system modeling. Two main aspects will be examined. First, simulation performance will be evaluated, since a performance gain is the main premise of abstract modeling. Second, the accuracy of the more abstract models will be examined. Weighting the speed benefits against the accuracy drawbacks allows the designer to decide on speed/accuracy trade-off applicable for a particular design stage.

4.1 Performance

The performance of each model has been measured in a scenario with two bus nodes: one acting as a master, one as a slave. A user transaction is transferred a constant number of times, without any delay in between. The simulation time (also referred as real time or wall clock time) for executing all repetitions of the user transaction was measured and the average execution time for a single user transaction was calculated. All tests have been performed on a Pentium 4, 2.8 GHz.

The results of performance measurements in terms of simulation time are shown in Figure 3. The x-axis denotes the size of a user transaction in

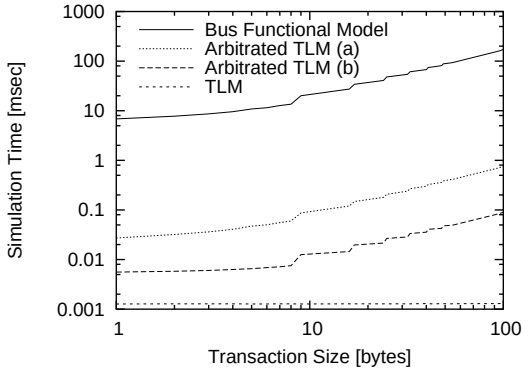

Figure 3. Simulation time

Feature	Bus Functional Model	ATLM (a)	ATLM (b)	TLM
simulation time [ms]	27.4	0.12	0.015	0.0012
simulation bandwidth [MByte/sec]	0.0006	0.127	1.05	12.3
speedup over bus functional model	1	228	1879	22124
speedup over next more accurate model	1	228	8	12

Table 2. Model performance comparison for sending 16 bytes.

bytes. The y-axis denotes the time the simulation spends for transferring of one user transaction. Table 2 compares the performance of the models for a 16 bytes user transaction.

The performance measurements confirm the expectations: the simulation speed increases with an increase of abstraction. The TLM model executes the fastest. Its execution time is independent of the transaction size, since a constant number of operations is executed for each transfer (one *memcpy* and one *waitfor*). Transferring 16 bytes (via multiple CAN messages) takes 0.0012 ms.

The next slower model is the ATLM (b), which does not model bit stuffing and CRC. Since the ATLM models the data transfer at the level of bus transactions (CAN messages), a step is noticeable in the graph for each 8 bytes - an additional CAN message is needed for transferring the user data. The execution time increases linearly with the amount of bus transactions. A 16 byte transaction is transfered in 0.015 ms.

The ATLM (a) performs 8 times slower than the ATLM (b), since it inspects every bit of the message for the bit stuffing and the CRC calculation. The effect of the additional effort can be seen by the increase of simulation time within one frame. Transferring a 16 bytes user transaction takes 0.12 ms.

The bus functional model is two orders of magnitude slower than the ATLM (a). The additional effort of serially transmitting the data and performing the bit synchronization requires more computing power. Additionally to the increased functionality, the structure of the implementation reduces the performance. For each bus node two additional threads of execution are required, one for the bit stream processor and one for the bit timing logic. It takes 27.4 ms for transferring 16 bytes.

4.2 Accuracy

In the previous section, the gain of speedup by using models at higher level of abstraction was quantified. Now we will evaluate, which accuracy limitations the designer has to accept for achieving the higher simulation speeds. However, unlike the performance measurements before, it is hard

to define a single expressive number that allows comparing the accuracy of the different models. The actual accuracy depends heavily on the environment and the actual application at hand.

4.2.1 Test Setup. A generic test setup with 4 bus nodes was used. Two nodes act as masters and two nodes act as slaves. During the test, each master transfers a predefined set of 5000 user transactions. The user transactions vary in message id, in length and content of the transaction (1 - 16 bytes) and in the delay between two transactions (simulating local computation). All varying parameters are linear random distributed. Each master sends from an exclusive range of message ids. One master will send messages with high priority ids (0-511), the other emits messages with low priority (ids 512-1023).

During the test execution, the start time and duration (each in simulated time) of each individual user transaction is recorded, separately for each master. The test is repeated once for each implemented bus model. Since the same set of user transactions is transferred by each model, their results are comparable and can be analyzed.

Bus contention is a major concern for bus usage in general. It is expected, that model accuracy varies significantly with bus contention. Therefore the described test was repeated for different bus contentions.

The bus contention can not be controlled directly in this test. Instead, the maximum delay between two user transactions of a master has been varied between test runs. Varying the maximum delay influences the bus utilization and, since two masters access the bus during the test, it correlates to the amount of bus contention. The actual amount of contention during was measured during test execution of the bus functional model.

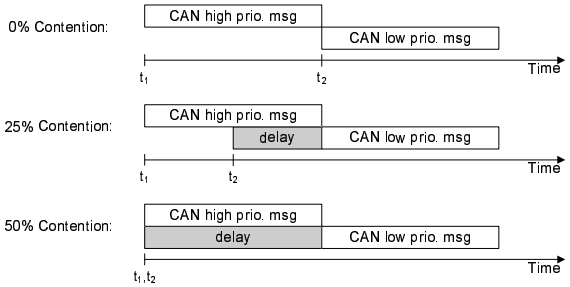

Figure 4. Example of bus contention.

For this paper, the contention is defined as the overlap between user transactions as shown in Figure 4. The actual amount of contention, as a result of a particular maximum transaction delay, has been measured with the bus functional model. For each CAN bit time, it was measured

whether one or two user transactions where active. A user transaction is active if the application is blocked for completion of the transaction. This definition of an active transaction is independent from the actual state of the bus node (e.g. pending, arbitration, active transmission). Given this basic definition, the contention is defined for this paper as:

$$contention = 100 * \frac{bus\ cycles\ with\ two\ active\ user\ transactions}{bus\ cycles\ with\ at\ least\ one\ active\ user\ transaction} \quad (1)$$

4.2.2 Analysis Based on Transfer Duration. As described above, a test run yields a execution record of each individual user transaction. The following paragraphs will describe the analysis of the measured data.

The transfer duration of an individual user transaction is an important measure for predicting the application latency due to bus access. Therefore, in a first step, the accuracy of the models has been evaluated with respect to the transfer duration. For this purpose, the error of an individual user transaction is defined as:

$$\begin{aligned} duration_{std}: &\quad transfer\ duration\ as\ per\ CAN\ standard \\ duration_{test}: &\quad transfer\ duration\ in\ model\ under\ test \\ error_i &= 100 * \frac{|duration_{test} - duration_{std}|}{duration_{std}} \end{aligned} \quad (2)$$

Given this error definition, a timing accurate model exhibits 0% error. It was avoided to directly express the accuracy in percent, since a particular model may have an error of more than 100% (i.e. the model under test predicts more than twice the simulated time).

The first set of graphs, Figure 5a for the high priority master and Figure 5b for the low priority master, show the average timing error for a user transaction for different amounts of bus contention.

Figure 5a shows that the ATLM (a), which includes bit stuffing and CRC calculation, performs as accurate as the bus functional model (both graphs lie on top of the x-axis). This result has to be seen in perspective to the restrictions of the test, which are: no propagation delay between sending and receiving on the CAN bus, all delays between user transactions are multiple of the CAN bit time and the test starts aligned to the bit clock of the first sender. With this restrictions, reasonable for a simulation environment only, all bus accesses are performed aligned to the CAN bit clock, and no sub cycle information is needed. In this situation the additional capabilities of the bus functional model, i.e. bit synchronization, are not exercised and both the bus functional model and the ATLM (a) perform with 100% accuracy.

The ATLM (b), due to the lack of modeling the bit stuffing and CRC, performs inaccurately. For messages in the high priority range, the in-

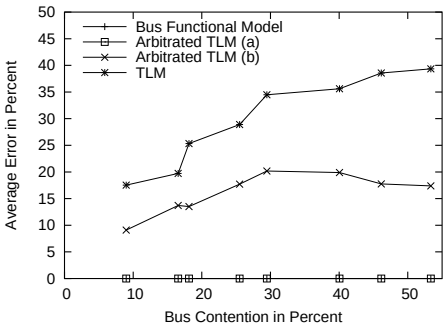

Figure 5a. Duration based error for high priority messages.

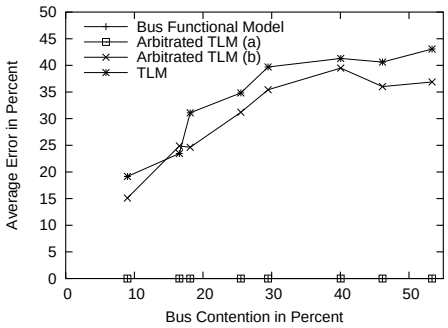

Figure 5b. Duration based error for low priority messages.

accuracy starts with 10% for low contention situations and platoes after linearly rising to 20% inaccuracy at 30% contention. With the lack of the bit stuff modeling, an individual message transfer is - depending on its content - shorter than in the bus functional model. Therefore, the arbitration interaction between the two senders differs. With an increasing contention the user transactions of the low priority band increasingly influence the high priority transactions. However an earlier started low priority transaction, which may consist of multiple CAN frames, can delay a later started high priority user transaction only for up to one frame. A second started CAN frame of the low priority transaction will lose arbitration, which leads to the plateau in inaccuracy at 30%.

Looking at the same scenario with reversed priorities, this limitation does not apply. A low priority user transaction may be delayed for a full high priority user transaction consisting of many CAN frames. Hence, the timing error of the ATLM (b) increases without a plateau for the low priority user transactions (Figure 5b) with increasing contention.

The TLM model, which simulates bus access on the level of user transactions only, both high and low priority give a uniform result. For both cases the inaccuracy increases with the bus contention. As to be expected, the TLM achieves the most inaccurate results (40% inaccuracy at 45% contention).

4.2.3 Analysis Based on Cumulative Transfer Duration.

The accuracy analysis based on the transfer duration is a measure to predict the application latency due to bus traffic. Additionally, the overall timing (e.g. when does the application finish?) is of interest for design decisions. For this, the same experimental results have been evaluated in terms of the cumulative transfer time, which is the sum of the user transaction durations. Figure 6a and Figure 6b show the results of the accuracy based on the cumulative transfer time.

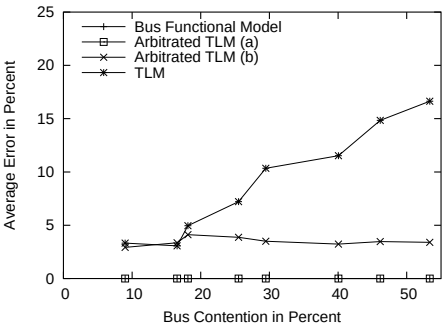

Figure 6a. Cumulative error for high priority messages.

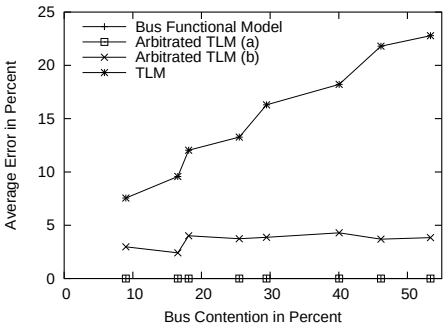

Figure 6b. Cumulative error for low priority messages.

The cumulative transfer time analysis reveals: mispredictions of the ATLM (b) for individual CAN frames average out during the test, since the model correctly captures arbitration. Regardless of priority a constant error of about 4% is measured. This can be attributed to not modeling the bit stuffing (which in average adds 4% bits). The TLM, with its coarse grain contention resolution independent of priority, shows for both priority ranges a linear increasing inaccuracy.

5. Conclusion

This paper has reported on a case study, based on the CAN bus, for abstract communication modeling. Three major models have been implemented: the bus functional model, the arbitrated transaction level model (ATLM) and the transaction level model (TLM). Additionally, two variances have been created for the ATLM.

The usability of the models has been evaluated. With respect to the simulation performance, a speedup of two magnitudes was measured from the bus functional model to the ATLM (a). With each further increase of abstraction (to the ATLM (b), and TLM) an additional speedup of one order of magnitude was measured.

A detailed analysis of the simulation accuracy of each model has been done. Based on the analysis results, Table 3 lists the fastest model, that yields acceptable results for a given environment and simulation focus.

Environment Condition	**Applicable Model**
• no overlap between masters bus access • early stage in design	TLM
• main focus on application finish time	ATLM (b)
• main focus on individual transfer delay	ATLM (a)
• synthesizable • using propagation delay	bus functional

Table 3. Model selection

The TLM can only be used in very early stages of the design. Its accuracy, for individual and cumulative transfer time, degrades heavily with increasing bus contention. The still fast ATLM (b) is applicable in scenarios, where the main focus is on the application finish time. This model is not suitable for predicting an individual transfer delay, since the duration based analysis has not shown acceptable results.

The ATLM (a), which includes bit stuffing and CRC calculation, has shown 100% accuracy given the test restrictions (e.g. no propagation delay). It is the fastest model that accurately predicts the delay of an individual transfer in all contention situations. The bus functional model is necessary as a synthesizable model, or in case the simulation includes propagation delay on the simulated CAN bus.

References

[1] Denny Brem and Dietmar Müller. Interface based system modeling of a CAN using SVE. In *EkompaSS Workshop*, Hanover, Germany, April 2003.

[2] M. Caldari et al. Transaction-level models for AMBA bus architecture using SystemC 2.0. In *DATE*, Munich, Germany, March 2003.

[3] Daniel D. Gajski et al. *SpecC: Specification Language and Design Methodology*. Kluwer Academic Publishers, 2000.

[4] A. Gerstlauer et al. System-Level Communication Modeling for Network-on-Chip Synthesis. In *ASP-DAC*, Shanghai, China, January 2005.

[5] A. Gerstlauer and D. Gajski. System-level abstraction semantics. In *ISSS*, Kyoto, Japan, October 2002.

[6] Thorsten Grötker, Stan Liao, Grant Martin, and Stuart Swan. *System Design with SystemC*. Kluwer Academic Publishers, 2002.

[7] Florian Hartwich and Armin Bassemir. The Configuration of the CAN Bit Timing. http://www.can.bosch.com/, 1999.

[8] Internation Organization for Standardization (ISO). *Reference Model of Open System Interconnection (OSI)*, second edition, 1994. ISO/IEC 7498 Standard.

[9] Philips. P8xC592: 8-bit microcontroller with on-chip CAN. http://www.semiconductors.philips.com, 1996.

[10] Robert Bosch GmbH. *CAN Specification*, 2.0 edition, 1991. http://www.can.bosch.com/.

[11] M. Sgroi et al. Addressing the system-on-a-chip interconnect woes through communication based design. In *DAC*, June 2001.

[12] R. Siegmund and D. Müller. SystemC^{SV}: An Extension of SystemC for Mixed Multi-Level Communication Modeling and Interface-Based System Design. In *DATE*, Munich, Germany, March 2001.