Message Sequence Charts Based Specification of the Communicating Threads to the Verified Image Compression Technique using Concurrent Wavelet Transform

Kamrul Hasan Talukder and Koichi Harada

***Abstract*— Message Sequence Charts (MSCs) can be used effectively to specify the communicating threads in the way where high-level transition systems is used to capture the control flow of the system components. This specification is amenable to formal verification. Here, we present the way how we can specify the communicating threads using MSCs and how we can verify the correctness of the concurrent wavelet transform used for the purpose of image compression. We have used Symbolic Model Verifier (SMV) for the use of formal verification. The introduction of wavelet transform for image compression followed by the MSC based specification for concurrent wavelet transform is presented. The concurrency in the threads for concurrent wavelet transform along with the grammar of the syntax formulated is accessed. At the end, the verification result of the concurrent wavelet transform using the SMV program is presented.**

***Index Terms*— Image Compression, Wavelet Transform, Message Sequence Charts, Symbolic Model Verifier (SMV), Formal Verification.**

I. Introduction

The display, storage and transportation of digital images have moved from obscurity to the commonplace in the last few decades. Now-a-days, many people interact with digital imagery, in one form or another, on a daily basis. The amount of data to be transported demands the development and improvement of the compression approaches. A certain amount of compression is possible without loss of information using *lossless* methods. However, many applications call for an order of magnitude more reduction of size than is possible to meet using such methods. Once the realm of *lossy* compression is entered, a trade-off in size vs. distortion must be made. The fundamental problem is to meet the competing goals of accurate approximation and reduction in storage requirements.

Compressing an image is significantly different from compressing raw binary data. Of course, general purpose compression programs can be used to compress images, but the result is less than optimal. This is because images have certain statistical properties which can be exploited by encoders specifically designed for them. Also, some of the finer details in the image can be sacrificed for the sake of saving a little more bandwidth or storage space.

The initial breakthrough in the compression of one-dimensional signals [1] was easily extended to the image domain by concatenating image rows or columns into a single stream. Some techniques such as Shannon Fano coding [2] and Huffman coding [3] [4] [5] [6] [7] [8] use redundancy-reduction mechanisms which result in shorter codes for more frequently appearing samples. It is necessary to scan the data samples in order to calculate their probabilities of occurrence and create an exact code. Adaptive variations of these techniques initially assume equal probability for all samples and calculate subsequent probability measures based on a fixed window length prior to the sample of interest. This allows local changes in probability measurements and achieves higher global compression. Run-length coding [9] is another redundancy-reduction coding method where in a scan-line each run of symbols is coded as a pair that specifies the symbol and the length of the run. While most redundancy-reduction methods are lossless, other arbitrarily lossy coding methods have achieved higher levels of compression. Transform coding [10], subband coding [11] [12] [13] [14], vector quantization [15] [16] [17] and predictive coding [18] [19] [20] [21] [22] [23] [24] [25] [26] [27] are among the ones that have achieved high levels of lossy compression. The major transform coding techniques include cosine/sine [28], Fourier [29], Hadamard [30], Haar [30] [10], slant [10] and principal-component (Karhunen-Loeve) transforms [31]. All transform coding based compression algorithms are pixel-based approaches which decompose a signal unto an orthonormal basis to achieve energy compaction. Lossy compression is then achieved by coding the high energy components and leaving out the low energy ones. While some transformations such as the Hadamard and Haar transforms can be performed relatively quickly, other transforms are computationally intensive and either require dedicated hardware or restrictions such as limiting the size of the transform to a power of two. Recently, second-generation compression algorithms based on human visual behavior [32] have been proposed which have the potential for much higher compression ratios.

In the recent years, the wavelet transform has emerged as a cutting edge technology within the field of image analysis.

Manuscript received February 12, 2008.

Kamrul Hasan Talukder is a PhD student in the Department of Information Engineering of Graduate School of Engineering, Hiroshima University, Japan. E-mail: khtalukder@hiroshima-u.ac.jp.

Koichi Harada is a Professor in the Department of Information Engineering of Graduate School of Engineering, Hiroshima University, Japan. E-mail: harada@cedar.mis.hiroshima-u.ac.jp.

The wavelet transformations have a wide variety of different applications in computer graphics including radiosity [33], multiresolution painting [34], curve design [35], mesh optimization [36], volume visualization [37], image searching [38] and one of the first applications in computer graphics, image compression. The Discrete Wavelet Transformation (DWT) provides adaptive spatial frequency resolution (better spatial resolution at high frequencies and better frequency resolution at low frequencies) that is well matched to the properties of an HVS.

However, the DWT is very computationally intensive process which requires innovative and computationally efficient method to obtain the image compression. The concurrent transformation might be a useful solution to this problem. In this paper, we investigate the concurrency in wavelet transformation for the compression of image. A verification of the system is also done.

Simulation and testing [39] are some of the traditional approaches for verifying the systems. Simulation and testing both involve making experiments before deploying the system in the field. While simulation is performed on an abstraction or a model of the system, testing is performed on the actual product. In both cases, these methods typically inject signals at certain points in the system and observe the resulting signals at other points. Checking all the possible interactions and finding potential pitfalls using simulation and testing techniques is not always possible. Formal verification [40], an appealing alternative to simulation and testing, conducts an exhaustive exploration of all possible behaviors of the system. Thus, when a design is marked correct by the formal method, it implies that all behaviors have been explored and the question of adequate coverage or a missed behavior becomes irrelevant. There are some robust tools for formal verification such as SMV, SPIN, COSPAN, VIS etc [40]. The method has been modeled in SMV and the properties of the system have been verified formally.

II. Image Compression and Wavelet transformation

Wavelet transform (WT) represents an image as a sum of wavelet functions (wavelets) with different locations and scales [41]. Any decomposition of an image into wavelets involves a pair of waveforms: one to characterize the high frequencies corresponding to the detailed parts of an image (wavelet function) and one for the low frequencies or smooth parts of an image (scaling function). Fig. 1 shows two waveforms of a family discovered in 1980 by Daubechies: the left one can be used to represent smooth parts of the image and the right one to represent detailed parts of the image. The two waveforms are translated and scaled on the time axis to generate a set of wavelet functions at different locations and on diverse scales. Each wavelet possesses the same number of cycles, such that, the wavelet gets longer while frequency reduces. Low frequencies are transformed with long functions (high scale). High frequencies are transformed with short functions (low scale). The analyzing wavelet is shifted over the full domain of the analyzed function. The result of WT is a set of wavelet coefficients, which measure the contribution of the wavelets at these locations and scales.

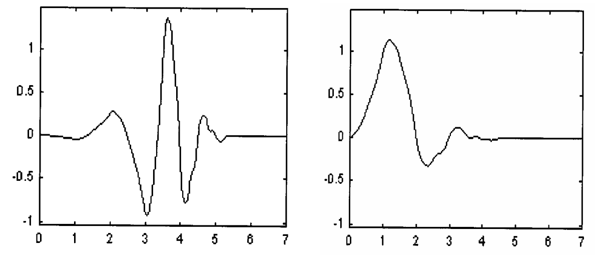


Fig. 1. Scaling and wavelet function.

WT performs multiresolution image analysis [42]. The result of multiresolution analysis is simultaneous image representation on different resolution levels [43]. The resolution is determined by the specified threshold below which all details are overlooked. The difference between two neighboring resolutions represents details. Therefore, an image can be represented by a low-resolution image (approximation or average part) and the details on each higher resolution level. Let us consider a one dimensional function $f(t)$. The approximation of the function $f(t)$ at the resolution level j is defined as $f_j(t)$. At the one level upper resolution $j+1$, the approximation of the function $f(t)$ is represented by $f_{j+1}(t)$. The details denoted by $d_j(t)$ are included in $f_{j+1}(t): f_{j+1}(t) = f_j(t) + d_j(t)$. This procedure is repetitive for several times and the function can be written as-

$$f(t) = f_j(t) + \sum_{k=j}^{n} d_k(t)$$

In the same way, the space of square integrable functions $L^2(R)$ can be treated as a composition of scaling subspaces V_j and wavelet subspaces W_j such that the approximation of $f(t)$ at resolution $j(f_j(t))$ is contained in V_j and the details $d_j(t)$ are in W_j. V_j and W_j are defined in terms of dilates and translates of scaling function Φ and wavelet function $\Psi : V_j = \{\Phi(2^j x - k) | k \in Z\}$ and $W_j = \{\Psi(2^j x - k) | k \in Z\}$. V_j and W_j are localized in scaled frequency *octaves* by the scale or resolution parameter 2^j and localized spatially by translation k. The scaling subspace V_j must be contained in all subspaces on higher resolutions ($V_j \subset V_{j+1}$). The wavelet subspaces W_j fill the gaps between successive scales: $V_{j+1} = V_j \oplus W_j$. We can start with an approximation on some scale V_0 and then use wavelets to fill in the missing details on finer scales. The finest resolution level includes all square integrable functions-

$$L^2(R) = V_0 + \bigoplus_{j=0}^{\infty} W_j$$

Since $\Phi \in V_0 \subset V_1$, it follows that the scaling function for multiresolution approximation can be found out as the solution to a two-scale dilational equation-

$$\Phi(x) = \sum_k a_L(k)\Phi(2x-k)$$

for some suitable sequence of coefficients $a_L(k)$. Once it has been found, an associated mother wavelet is given by a similar looking formula-

$$\Psi(x) = \sum_k a_H(k)\Phi(2x-k)$$

One of the big discoveries for wavelet analysis was that perfect reconstruction filter banks could be formed using the coefficient sequences $a_L(k)$ and $a_H(k)$ as shown in Fig. 2. The input sequence x is convolved with high-pass (HPF) and low-pass (LPF) filters $a_H(k)$ and $a_L(k)$ and each result is downsampled by two, yielding the transform signals x_H and x_L. The signal is reconstructed through upsampling and convolution with high and low synthesis filters $s_H(k)$ and $s_L(k)$. For properly designed filters, the signal x is reconstructed exactly ($y = x$).

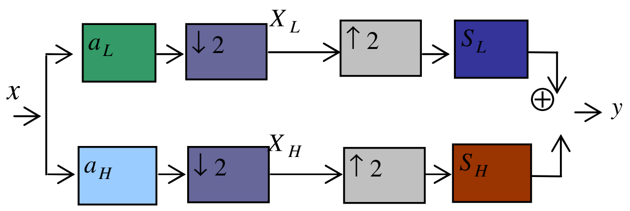


Fig. 2. Two-channel Filter Bank

The choice of filter not only decides if the perfect reconstruction is possible or not, it also determines the shape of wavelet to be used to perform the analysis. By cascading the analysis filter bank with itself several times, a digital signal decomposition with dyadic frequency scaling known as DWT can be formed. An efficient way to implement this scheme using filters was developed by Mallat [43]. The new twist that wavelets bring to filter banks is connection between multiresolution analysis and digital signal processing performed on discrete, sampled signals.

III. Concurrency in WT for Image Compression

We know that wavelet transformation entails transformation of image data horizontally first and then vertically. Here we divide the image plane into *n* horizontal sections which are horizontally transformed *concurrently*. After then the image is divided into *n* vertical sections which are then vertically transformed *concurrently*. It is not a must that the number of horizontal sections is equal to the number of vertical sections.

But the problem lies in the concurrency. The system just proposed lets the possibility for vertical transformation to begin on some vertical sections before horizontal transformation in all sections is completed. Vertical sections that are already horizontally transformed can be vertically transformed. That allows the possibility for threads that completed horizontal transformation to go on to vertical transformation without having to wait on other threads to complete horizontal transformation. Before a vertical section is available for transformation, one condition that must be met is that all horizontal sections transform *n* size data horizontally such that an *n* wide vertical section is available with all data points already horizontally transformed. The following figure 3 shows the control flow of a single thread.

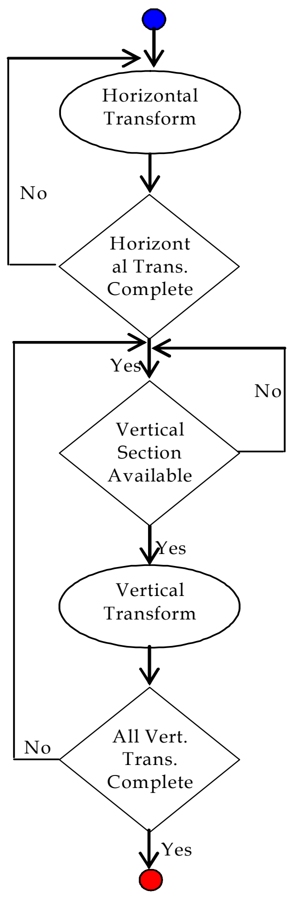

Horizontal Transform
No
Horizontal Trans. Complete
Yes
No
Vertical Section Available
Yes
Vertical Transform
No
All Vert. Trans. Complete
Yes

Fig. 3. Control flow of a single thread

The assertion for the verification is that at any time, the vertical transformation does not start on a vertical section that is not horizontally transformed.

A. *Communication among threads*

A message passing library providing two levels of abstraction namely channel and topology has been developed, for the communication that occurs among threads. These message passing classes can be the part of a larger system that provides a class library for threads, thread synchronization, and message passing. We have used the Message Sequence Charts (MSCs) to visualize the interactions among the threads.

B. *Message Sequence Charts (MSCs)*

Message Sequence Charts (MSCs) are an attractive visual formalism that is often used in the early stage of system design to specify the system requirements. A main advantage of an MSC is its clear graphical layout which immediately gives an intuitive understanding of the described system behavior [44]. MSCs are particularly suited to describe the distributed telecommunication software [45] [46]. The wide ranges of use of MSCs are usually in the distributed systems and in a number of software methodologies [46] [47] [48]. In a distributed system, MSCs mainly concentrate on the exchange of messages among various processes and their environments as well as some internal actions in these processes. MSCs are also known as object interaction diagrams, timing sequence diagrams and message flow diagrams.

C. Modeling the Communications

In order to establish the communication between multiple threads, we require some work to set up and maintain the communication channels. There are several ways to communicate between threads, with some being more efficient than others.

One of the simplest ways to communicate state information between threads is to use a shared object or shared block of memory. A shared object requires very little setup—all we have to do is make sure each thread has a pointer to the object. The object contains whatever custom information we need to communicate between threads, so it should be very efficient.

The second option is the port-based communication. Ports offer a fast and reliable way to communicate between threads and processes on the same or different computers. Ports are also a fairly standard form of communication on many different platforms and their use is well established. In Mac OS X, a port implementation is provided by the Mach kernel. These Mach ports can be used to pass data between processes on the same computer.

The third way is the use of the message queues. The message queues offer an easy-to-use abstraction for thread communication. A message queue is a first-in, first-out (FIFO) queue that manages incoming and outgoing data for the thread. A thread can have both an input and an output queue. The input queue contains work the thread needs to perform, while the output queue contains the results of that work.

To establish communication between the threads, we model it as a queue of message, which is the integral part of the threads. The following Fig. 4 shows the modeling of the channel as a queue of message from thread *i* to thread *i*+1. The message is pushed through the *tail* of the queue from the thread *i* side and the message is received from the *head* of the queue at the thread *i*+1 side. Fig. 5 shows the modeling of the communication channel as a queue for the message from thread *i*+1 to thread *i*. The message is sent from the thread *i*+1 side and it is received at the *head* of the queue at the thread *i* side.

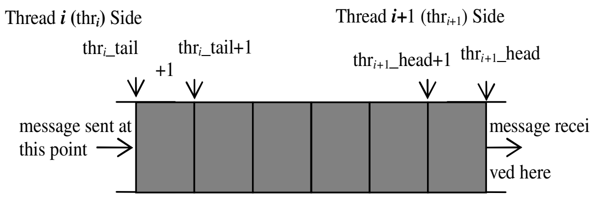

Thread **i** (thr$_i$) Side
Thread **i**+1 (thr$_{i+1}$) Side
thr$_i$_tail
thr$_i$_tail+1
+1
thr$_{i+1}$_head+1
thr$_{i+1}$_head
message sent at this point
message recei ved here

Fig. 4 Channel for message from thread *i* to thread *i*+1

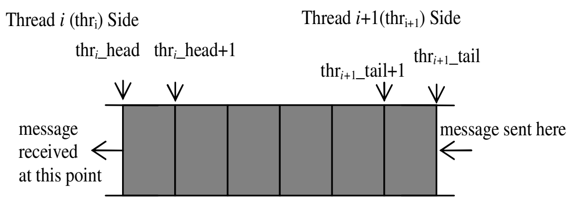

Thread *i* (thr$_i$) Side
Thread *i*+1(thr$_{i+1}$) Side
thr$_i$_head
thr$_i$_head+1
thr$_{i+1}$_tail+1
thr$_{i+1}$_tail
message received at this point
message sent here

Fig. 5 Channel for reply from thread *i*+1 to thread *i*

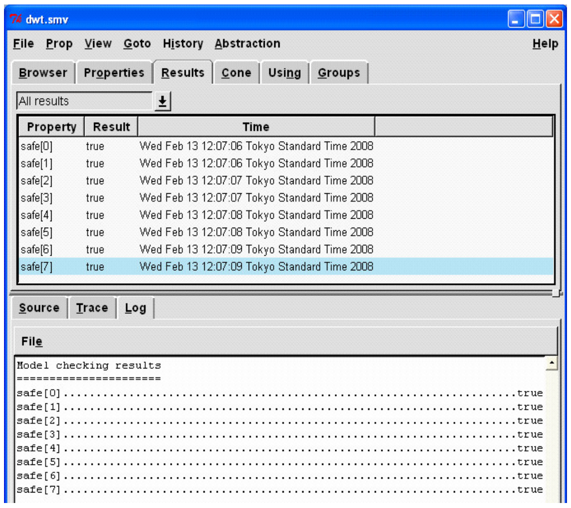
The communication among threads for a communication system where different threads take part in can be specified by a grammar. The syntax for it is formulated and is tabulated in Table 1 in the appendix.

D. Verification

We use the SMV [49] as the verification tool. Here the number of threads is defined as SIZE = 8. Using the loop from 0 to 7 we verify if any of the threads starts vertical transformation while some other thread(s) is/are still on the transformation of the horizontal part of the same section (unsafe). It is found that each of the threads is in the safe state (desired) all the time. It means that in all states of the transition system it is true that no vertical transformation gets started until the all the horizontal section is completed. This specification is the most important one that we must get true. The part of the SMV code is shown in Fig. 6 and the snapshot of the verification result is shown in Fig. 7.

```
for (index=0;index<SIZE;index=index+1){
    loc[index],idle[index]:boolean;
    idle[index]:=free & freed=index;
    loc[index]:=alloc & pos_ack & allocd=index;
    safe[index]:SPEC AG(A(~idle[index] U loc[index]));
}
```

Fig.6 Part of the SMV Code


dwt.smv
File Prop View Goto History Abstraction Help
Browser Properties Results Cone Using Groups
All results
Source Trace Log
File

Property	Result	Time
safe[0]	true	Wed Feb 13 12:07:06 Tokyo Standard Time 2008
safe[1]	true	Wed Feb 13 12:07:06 Tokyo Standard Time 2008
safe[2]	true	Wed Feb 13 12:07:07 Tokyo Standard Time 2008
safe[3]	true	Wed Feb 13 12:07:07 Tokyo Standard Time 2008
safe[4]	true	Wed Feb 13 12:07:08 Tokyo Standard Time 2008
safe[5]	true	Wed Feb 13 12:07:08 Tokyo Standard Time 2008
safe[6]	true	Wed Feb 13 12:07:09 Tokyo Standard Time 2008
safe[7]	true	Wed Feb 13 12:07:09 Tokyo Standard Time 2008

```
Model checking results
======================
safe[0]..................................................................true
safe[1]..................................................................true
safe[2]..................................................................true
safe[3]..................................................................true
safe[4]..................................................................true
safe[5]..................................................................true
safe[6]..................................................................true
safe[7]..................................................................true
```

Fig. 7 Verification Results

IV. CONCLUSION

The way to specify the communicating threads using MSCs and the verification of the correctness of the concurrent wavelet transform used for image compression are presented. We have used Symbolic Model Verifier (SMV)) for the use of formal verification. The concurrency in the threads for concurrent wavelet transform along with the grammar of the syntax formulated is discussed and presented here. We find that all the communicating threads are in the safe states all the time. One of the problems associated in the current version of the work is, in SMV, the smaller size of the queue shared by different threads. Another bottleneck is the smaller number of participating threads.

REFERENCES

[1] R. Williams. *Adaptive Data Compression*. Kluwer Academic Publishers, 1991.
[2] C. Shannon. A mathematical theory of communication. *Bell*

Systems Technical Journal, 27:623-656, 1948.

[3] N. Faller. An adaptive system for data compression. In *Proceedings of the Asilomar Conference on Circuits, Systems and Computers*, pages 593-597, 1973.

[4] R. Gallager. Variations on a theme by Huffman. *IEEE Transactions on Information Theory*, 24:668-674, 1978.

[5] D. Knuth. Optimal binary search trees. *Acta Informatica*, 1:14-25, 1971.

[6] J. Vitter. Design and analysis of dynamic Huffman codes. *Journal of the Association for Computing Machinary*, 34:825-845, 1987.

[7] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23:337-343, 1977.

[8] J. Ziv and A. Lempel. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, 24:530-536, 1978.

[9] T. Huang. Run length coding and its extensions. In T. Huang and O. Tretiak, editors, *Picture Bandwidth Compression*, pages 231-264. Gordon and Breach, 1972.

[10] A. Jain. *Fundamentals of Digital Image Processing*. Prentice-Hall, 1989.

[11] J. Lim. *Two-Dimensional Signal and Image Processing*. Prentice-Hall, 1990.

[12] D. O'Shaughnessy. *Speech Communication: Human and Machine*. Addison-Wesley, 1987.

[13] M. Vetterli. Multi-dimensional sub-band coding: Some theory and algorithms. *Signal Processing*, 6:97-112, 1984.

[14] J. Woods and S. O'Niel. Subband coding of images. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 34:1278-1288, 1986.

[15] J. Lim. *Two-Dimensional Signal and Image Processing*. Prentice-Hall, 1990.

[16] Y. Linde, A. Buzo, and R. Gray. An algorithm for vector quantizer design. *IEEE Transactions on Communications*, 28:84-95, 1980.

[17] J. Makhoul, S. Roucos, and H. Gish. Vector quantization in speech coding. *Proceedings of the IEEE*, 73:1551-1558, 1985.

[18] J. Abate. Linear adaptive delta modulation. *Proceedings of the IEEE*, 55:298-308, 1967.

[19] S. Alexander and S. Rajala. Image compression results using LMS adaptive algorithm. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 33:712-717, 1985.

[20] C. Cutler. Differential quantization of communication signals. U.S. Patent 2 605 361, 1952.

[21] A. Habibi. Survey of adaptive image coding techniques. *IEEE Transactions on Communications*, 25:1275-1284, 1977.

[22] N. Jayant. Adaptive delta modulation with a one-bit memory. *Bell Systems Technical Journal*, 49:321-343, 1970.

[23] A. Kolmogorov. Interpolation and extrapolation of stationary random series. *Journal of the Soviet Academy of Science*, pages 3-14, 1941.

[24] T. Lei, N. Scheinberg, and D. Schilling. Adaptive delta modulation system for video encoding. *IEEE Transactions on Communications*, 25:1302-1314, 1977.

[25] J. O'Neal. Predictive quantization system (differential pulse code modulation) for the transmission of television signals. *Bell Systems Technical Journal*, 45:689-721, 1966.

[26] P. Pirsch. Adaptive intra/interframe DPCM coder. *Bell Systems Technical Journal*, 61:747-764, 1982.

[27] C. Song, J. Garondick, and D. Schilling. A variable-step-size robust delta modulator. *IEEE Transactions on Communications*, 19:1033-1044, 1971.

[28] N. Ahmed, T. Natarajan, and K. Rao. Discrete cosine transform. *IEEE Transactions on Computers*, 23:90-93, 1974.

[29] G. Anderson and T. Huang. Piecewise Fourier transformation for picture bandwidth compression. *IEEE Transactions on Communications*, 19:133-140, 1971.

[30] B. Fino. Relations between Haar and Walsh/Hadamard transforms. *Proceedings of the IEEE*, 60:647-648, 1972.

[31] H. Andrews. *Computer Techniques in Image Processing*. Academic Press, 1970.

[32] A. Mazzarri and R. Leonardi. Perceptual embedded image coding using wavelet transforms. In *Proceedings of the International Conference on Image Processing*, volume I, pages 586-587, 1995.

[33] Gortler. S., Schröder, P., Cohen, M., and Hanrahan, P., "Wavelet Radiosity", in Proc. SIGGRAPH, 1993, pp. 221-230.

[34] Berman, D., Bartell, J. and Salesin, D., "Multiresolution Painting and Compositing", in Proc. SIGGRAPH, 1994, pp. 85-90.

[35] Finkelstein. A. and Salesin, D., "Multiresolution Curves", in Proc. SIGGRAPH, 1994, pp. 261-268.

[36] Eck, M., DeRose, T., Duchamp, T., Hoppe, H., Lounsberry, M. and Stuetzle, W., "Multiresolution Analysis of Arbitrary Meshes", in Proc. SIGGRAPH, 1995, pp. 173-182.

[37] Lippert, L. and Gross, M., "Fast Wavelet Based Volume Rendering by Accumulation of Transparent Texture Maps", in Proc. EUROGRAPHICS, 1995, pp. 431-443.

[38] Jacobs, C., Finkelstein, A. and Salesin, D., "Fast Multiresolution Image Querying", in Proc. SIGGRAPH, 1995, pp. 277-286.

[39] Myers, Glenford J. “The Art of Software Testing”, John Wiley and Sons. ISBN 0-471-04328-1, 1979.

[40] Edmund M. Clakre, Jr. Oma FrumBerg and Doron A. Paled, “Model Checking”, The MIT Press, Second Printing, 2000.

[41] Eric J. Stollnitz, Tony D. Derose and David H. Salesin, “Wavelets for Computer Graphics”, Morgan Kaufmann Publishers, Inc., San Francisco.

[42] S. Mallat, “A theory of multiresolution signal decomposition: The wavelet representation,” *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 11, pp. 674–693, July 1989.

[43] S. Mallat, “Multifrequency channel decomposition of images and wavelet models,” *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 37, pp. 2091-2110, 1989.

[44] Ekkart Rudolph, Peter Graubmann and Jens Grabowski: *Tutorial on Message Sequence Charts.* MSC Tutorial of the 7th SDL Forum, September 1995, Norway.

[45] ITU-TS Recommendation Z.120: *Message Sequence Chart (MSC).* ITU-TS, Geneva (1997).

[46] Rudolph, E., Graubmann, P. and Grabowski, J.: *Tutorial on Message Sequence Charts*. In Computer Networks and ISDN Systems-SDL and MSC, Volume 28 (1996).

[47] Booch, G., Jacobson, I. and Rumbaugh, J.,: *Unified Modeling Language User Guide*. Addison-Wesley (1997).

[48] Harel, D. and Gery, E.: *Executable object modeling with state charts*. IEEE Computers, July 1997, pp. 31-42.

[49] Cadence Berkeley Laboratories, Free download from the website: http://www.kenmcmil.com/smv.html, California, USA. SMV Model Checker, 1999.

APPENDIX

The syntax

S	→ **THREAD thread-name {** thr-dec-part trans-sch-dec-part **}**
thr-dec-part	→ thr-dec-part thr-dec | thr-dec
thr-dec	→ **PROCESS process-name {**var-dec-part equation-part **}**
type	→ basic-type | enum-type | array-type
basic-type	→ range-type | **BOOLEAN**
enum-type	→ { id-list }
id-list	→ **identifier (, identifier)**$^+$
array-type	→ **ARRAY** range-type **OF** basic-type
range-type	→ **interger-const .. interger-const**
var-dec-part	→ var-dec-part var-dec**;** | ε
var-dec	→ **type-id :** id-list
equation-part	→ **EQUATION** id-list**;**
trans-sch-dec-part	→ trans-sch-dec trans-sch-dec-part | trans-sch-dec
trans-sch-dec	→ **SCHEME trans-schm-name {** trans-list **}**
trans-list	→ trans-dec trans-list | trans-dec
trans-dec	→ **TRANSACTION trans-name { AGENTS {** agent-list **}** condition-section **; }** | **TRANSACTION trans-name { AGENTS {** agent-list **}**
condition-section	→ **CONDITION** condition **;**
agent-list	→ agent-list agent | agent
agent	→ **process-name :** event-list
event-list	→ event **,** event-list | event **;**
event	→ **send(mesg-id, var)** | **send(mesg-id, const, type)** | **recv(process-name.mesg-id)** | {action}
action-atom	→ simple-stmt **;** | if-stmt
action	→ action action-atom | action-atom
simple-stmt	→ **var :=** expr | **var :=DIN**
expr	→ expr [* | / | &] F | F
F	→ F **+** G | F **-** G | F | G
G	→ G **mod** H | H
H	→ H **RelOp** I | I
I	→ ~I | -I | (expr) | var | const
const	→ **integer-const** | **boolean-const**
condition	→ condition & condition-atom | condition-atom | (condition-atom)
condition-atom	→ ~prop | prop
prop	→ prop **or** prop-atom | prop-atom
prop-atom	→ scoped-var relop const | scoped-var relop scoped-var | scoped-var
relop	→ = | < | > | ≤ | ≥ | !=
if-stmt	→ **IF** expr { action } | **IF** expr action-atom | **IF** expr {action}**ELSE** {action}
var	→ **identifier** | **identifier**[**identifier**] | **identifier** [**integer-const**]
scoped-var	→ **process-name.**var
thread-name	→ **identifier**
process-name	→ **identifier**
trans-name	→ **identifier**
mesg-id	→ **identifier**