Formal Verification of Temporal Specifications for Compile-Time Error Detection*

Seongbae Eun[1] and Byeong Man Kim[2]
[1]Center for Artificial Intelligence Research
Department of Information Communication Engineering
HanNam University
133 Ojung-Dong, Deaduk-Gu, Taejon 300-791, Korea
[2]Department of Computer Engineering
Kumoh National University of Technology
188 Shinpyong-Dong, Kumi, Korea
e-mail) sbeun@hudice.hannam.ac.kr

keyword: Interactive Multimedia Application, Authoring System, Compile-time Error Detection, Calculus of Communicating System, Formal Specification and Verification

1 Introduction

From the early days of computer systems, a lot of endeavors have been devoted in developing interactive computer systems for improving the quality of communication between human and human, or human and computers. As the analogue interactive videodiscs combined with computer graphics and display technology have made the area of interactive applications enlarge, Interactive Multimedia Applications(IMAs) have been emerged as a hot spot of trends in the field. In the late 1980s, digital video and digital audio supported in such systems as DVI carried one step further the capabilities of IMAs.

It has been one of the key issues in the field that those who want to build IMAs are not professional computer programmers but educators and nonprogrammers who are not skilled to program in conventional programming languages. In the past, for developing CAI(Computer Aided Instruction) courseware which is a typical example of IMAs, courseware developers should have programmed in a procedural language like Pascal, C, and COBOL[1]. This took the design of coursewares away from educators and into the hands of programmers. For the ease of building IMAs, authoring systems have been developed, which provide educators and nonprogrammers with the tools to write CAI courses in plain English or visual programming environments rather than conventional programming languages[2].

Although authoring systems have helped authors to build IMAs, we think building IMAs gets difficult as much as IMAs get more complex[3]. Let us imagine the process of building

*This work is supported in part by Center for Artificial Intelligence Research

an multimedia encyclopedia, A lot of nodes containing text, sound, and motion video are linked to each other for navigation and the presentation of nodes should be synchronized between them temporally and spatially. A lot of navigation and synchronization links make the specification complex and confused, which may lead to the errors of specification. In commercial authoring systems like ToolBook, Authorware, and etc., the correctness of the specification mainly relies on human's perception, that is, by only watching the presentation. In Little's Timed Petri Net(TPN)[4] that specifies temporal synchronizations in IMA formally, the temporal correctness of a specification can be checked automatically. Given a specification, the presentation times of related nodes are sum up and the summation is compared to the total presentation time. If the summation is greater than the total presentation time, it means that the specification is wrong. But, Little's TPN does not consider the interactions with users, so it is not appropriate to build complex IMA.

We have already developed an authoring system called Eventor[5] based on a formal specification mechanism, Calculus of Communicating System(CCS). But, we think that it is not sufficient to handle complex IMAs efficiently. Now, we intend to extend Eventor by providing with a tool to detect errors in specifications at compile-time. For example, an author describes an IMA with Eventor, a set of CCS notations is derived automatically, and then, the correctness of specifications is verified via the formal verification mechanism of CCS. In this paper, we describe the requirements of advanced authoring systems and verification mechanism that is essential for the error detecting tool.

2 Requirements for Advanced Authoring Systems

We investigate the requirements of authoring systems capable of manipulating such complex IMAs as follows.

- They should provide authors with *divide and conquer*(D&C) paradigm. D&C paradigm is a traditional and intuitive paradigm employed to solve large and complex problems in various fields. In the case of creating IMAs, the editing scenario based on D&C paradigm looks like this quotation.

  > A CAI application is composed of three parts presented serially, the first component spends about one minute presenting a video and sounds contemporaneously, which explains an introduction of the CAI application. The second part also consists of five components, two of which are presented parallelly and one is selected among the others by user's choice after the two are completed. And, $\cdots$.

- They should specify formally the properties inherent in IMAs such as temporal, spatial synchronizations, and interactivity with users. *Formal Specification* can get rid of ambiguity hidden in programmers' specifications and make clear the syntax and semantics. Besides, it can provide authoring systems with the capability to check syntactic correctness of visual expressions or semantic anomalies in them as LOTOS[6] has been developed for specifying the behaviors of distributed systems.

- They should provide authors with *automatic aid* tools like validation of temporal constraints, verification of visual expressions to check the consistency between two visual expressions when D&C paradigm is employed, and automatic temporal layout[7] to help authors reduce burdens of specifying and verifying the details of temporal informations.

Our earlier system is called Eventor which is the compound word of two words, *event* and *editor*. Eventor consists of three individual tools, Temporal Synchronizer, Spatial Synchronizer, and User Interaction Builder. In Eventor, the specification is carried out via D&C approach. For example, a slide show is designed roughly to be two composite objects, **A** and **B**. **A** represents slides and **B** represents their speech annotations. Next, each composite object is designed and described in more detail. In our system, a detailed specification for a composite object can be verified to be equivalent to the specification of the composite object itself via the formal verification mechanism facilitated in CCS, which can detect the specification errors inherent in the detailed specification.

In the original CCS, although the verification mechanism has been well established for conventional distributed applications[6], it should be supplemented for dealing with complex IMAs. One of supplements is to specify and verify temporal constraints explicitly in its verification mechanism. Hansson's Timed Probabilistic CCS(TPCCS) is one of the studies to extend CCS with explicit temporal constraints[8] but it is rather restrictive in that it does not provide the description of the latest time of an event. Another is to isolate the specification errors from system malfunctions caused by the lack of system resources like CPUs, memories, bus bandwidth and so on. If a system is heavily loaded, the quality of presentation is not preserved. What matters is to identify the malfunction as wrong specifications. Hence, it is crucial to analyze the resources required for the presentation of an IMA at compile-time and warn the possibility of malfunction due to the lack of resources. It contains scheduling resources in a system and check whether the resources are sufficient or not.

3 Formal Verification of Temporal Constraints

In this section, we introduce the notation and the semantics of Temporal CCS(TCCS) that is the extension of CCS with explicit temporal constraints. We also suggest an example that the verification is used for detecting specification errors.

In CCS, each object has input ports denoted as **a?** and output ports, **a!**, where **a** is the identifier of the port. If an object, A, has an input port, a?, and an output port, b!, and its notation is A = a?.b!.A, then it means that the object starts to make action as soon as it receives a message from the input port, a?, and delivers a message via the output port, b! after the action. We propose TCCS in which CCS notations are annotated with temporal constraints, which consists of two times, the earlist time and the latest time. For example, a![n,n] means that the message a can be transmitted after n unit times pass. a![0,0] also means that it should be output without time delay. a![0,∞] means that there is no timing constraint. Generally speaking, A = a![n,m]Á means that object A is able to send message a within n time unit and m time unit spent from its initial state.

Figure 1 shows an example described in the form of TCCS, which represents the presentation of a slide show. In the show, some timing constraints are described. That is, a slide stops its

presentation after 3 time units and the next slide is presented automatically. If a user wants, the next slide can be presented via the order of the user before 3 time unit. The next slide can not be presented before 1 time unit. A background music starts and stops its presentation at the same time that the slide show does. The total presentation time of the show is limited within 15 unit times. In the figure, the slide show is designed in the top down approach. Figure 1-1)

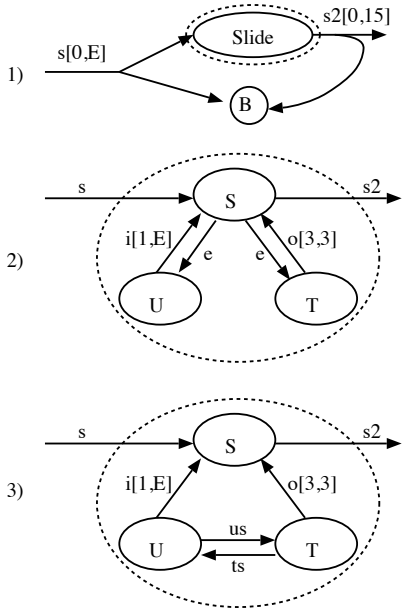

Figure 1: Specification of Slide Presentation using TCCS

is the top level description of the slide show, where a composite object depicted with a dotted ellipse, **Slide**, represents the main part of slide show, and **B** is an atomic object to represent the presentation of the background music. **s2[0,15]** represents the timing constraint that the show should stop within 15 time units. Figure 1-2) shows the detailed design of the composite object, **Slide**. In 2), **S** tasks a role of presenting the slides, and **U** tasks a role of suspending the presentation of the current slide according to user's command. **T** resumes the presentation of the next slide at every 3 time units. If the start message, **s**, arrives, **S** starts the presentation of a slide and sends **e** message to **U** and **T** objects. **U** inputs user's suspend command and sends **i** message to **S** after 1 time unit. **T** also sends **o** message after 3 time units. If **S** object receives either **i** or **o**, then it presents the next slide. At that time, it reinitializes **U** and **T** via sending **e** messgae to both objects.

Figure 1-3) shows a detailed specification as it carries out the same function but it contains specification error, which can be found automatically by formal verification technique of CCS. In CCS, two specifications are verified to be equivalent and we intend to detect specification errors at compile-time via showing that a detailed specification of a composite object is equivalent to its rough specification. For example, Figure 1-1) is a rough specification of a slide show and 2) and 3) are its detailed specifications. In the case of 1) and 2), the equivalence of two specifications will hold but it won't in 1) and 3). Then, we can conclude that the specification of 3) has some errors.

4 Summary and Further Work

We are going to develop an authoring tool or language that describe complex IMAs efficiently. It adopts top-down approach to divide complex IMAs into simpler modules and each module is conquered in turn. It is also equipped with the function of compile-time error detection, which is based on the formal specification mechanism, TCCS and its verification mechanism. The primary part of specification and verification has been designed. Now, we design formally the details of TCCS and the verification mechanism for TCCS. We think that it will be completed soon. We will implement the verifier till the end of this year.

As a further research, we have a plan to develop a script language to facilitate the compile-time error detection, which consists of two main functions. One is to notice the syntactic anomaly and the other is to schedule resources required for the presentation of an application and to inform whether they are sufficient or which one is a bottleneck.

References

[1] S. Ambron and K.(eds) Hooper. *Learing with Interactive Multimedia*. Microsoft Press, Redmond, WA, 1990. ISBN 1-55615-282-5.

[2] C. J. Anderson and M. D. Veljkov. *Creating Interactive Multimedia : A Practical Guide*. Scott, Foresman and Company, Glenview, Illinois, London. ISBN 0-673-46141-6.

[3] L. Hardman, G. Rossum, and D.C.A. Bulterman. Structured multimedia authoring. In *Proc. of ACM Multimedia'93*, pages 167–173, Aug. 1993.

[4] T. D. C. Little, A. Ghafoor, and C. Y. R. Chen. Conceptual data models for time-dependent multimedia data. In *Proc. of Multimedia Information systems*, pages 86–110, Tempe, Arizona, Feb. 1992.

[5] S. Eun, E.S. No, H.C. Kim, H. Yoon, and S.R. Maeng. Eventor: An authoring system for interactive multimedia applications. *ACM Multimedia Systems*, 2(3), 1994.

[6] ISO/IEC. Information retrieval, transfer and management for osi - lotos - a formal description technique based on the temporal ordering of observable behaviour. International standard, iso-8807, Sep. 1987.

[7] M.C. Buchanan and P.T. Zellweger. Automatic temporal layout mechanisms. In *Proc. of ACM Multimedia'93*, pages 341–350, Aug. 1993.

[8] H. Hansson and B. Jonsson. A calculus for communicating systems with time and probabilities. In *Proceedings of Real-Time Systems Symposium*, pages 278–286, 1990.