

Lecture - 32
Modeling and Specification – II

(Refer Slide Time: 00:30)

Requirements for specification & modeling techniques (5)

- Presence of programming elements
- Executability (no algebraic specification)
- Support for the design of large systems (OO)
- Domain-specific support
- Readability
- Portability and flexibility
- Termination
- Support for non-standard I/O devices
- Non-functional properties
- Support for the design of dependable systems
- No obstacles for efficient implementation
- Adequate model of computation

What does it mean "to compute"?

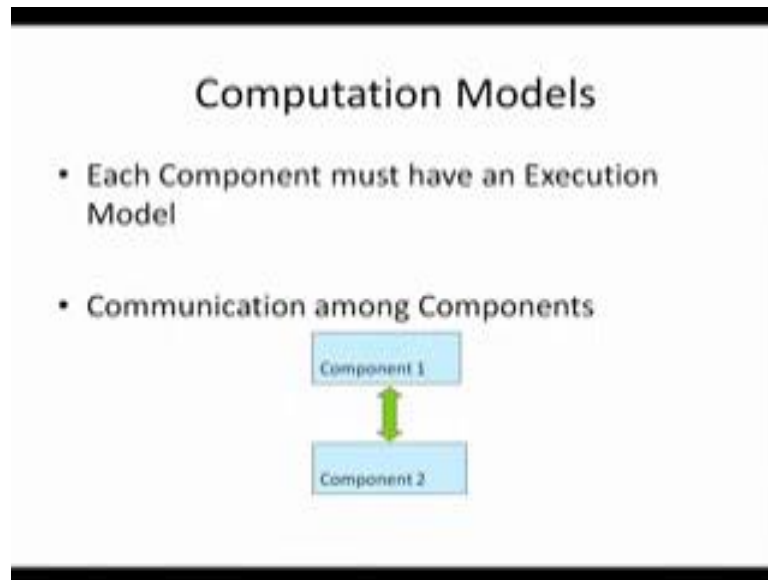
tu technische universität dortmund fakultät für informatik © P. Marwedel, informatik 12, 2012 - 9 -

Good morning! So, we started discussing on different models for specification. And, in the last lecture we came up to this point that specification and modeling language or specification and modeling method will require; it is desirable to have some features like presence of programming elements, which will enable you to specify the behaviour in an elegant manner and unambiguous manner. The specification must be executable, so that we can find out whether the specification actually does what we intended. It must be readable.

There will be other non-functional properties. Non-functional properties like for example, the timing constraint. Timing constraint is not the functional property. A particular thing has to be done. Now that has to be done within this particular time. That is an additional constraint that I am imposing on the behaviour. The same behaviour could be little relaxed for some other application. So, those are; the functions were the same. So, these are the non-functional properties.

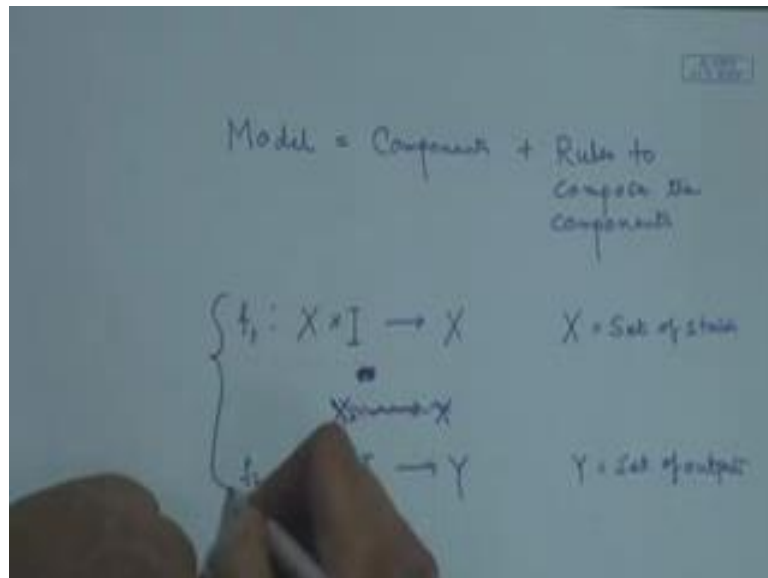
Now, we must have adequate model of computation. That is where we stopped in the last class. Now, what does it mean to compute? When you say we need adequate model of representing the computation in an embedded system, what do you mean?

(Refer Slide Time: 02:10)



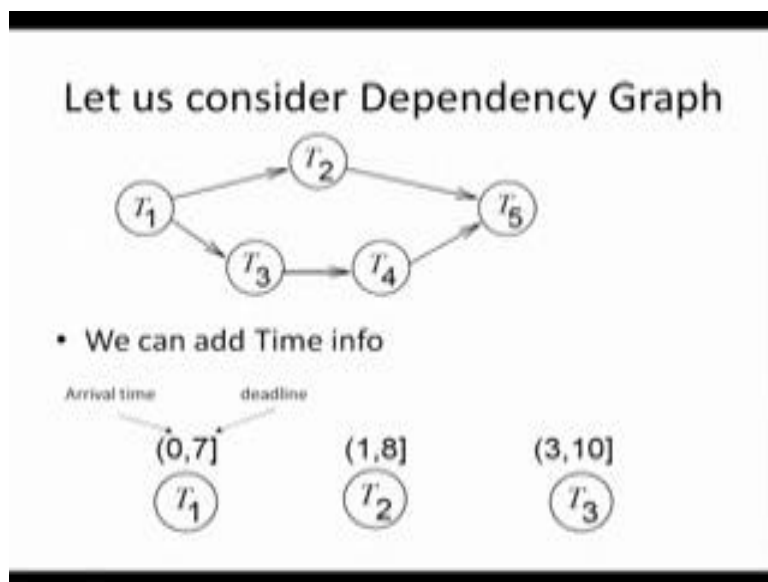
Actually, the computation models, each computation must have an execution model. That must be that this particular behaviour of the computation; must be executable. Now that computation should be; one thing is that it should be dividable into different components. The whole thing; will the whole specification. Since, you were talking of the hierarchy and all those, the whole system can be visualized as comprising a number of components. Now, each of this components will have their own behaviour. And, there should be a communication model for each of this components, that we have to specify. So, any computation model must have two things; the description of each component and how these components are combined. In other words, I can say that a model consists of a set of rules by which I can compose the component.

(Refer Slide Time: 03:26)



So, models will be; I can just say that a model will be components plus rules to compose the components. So, there are two things. One is the behaviour of the components and how they connect. How they connect means again how they communicate.

(Refer Slide Time: 04:07)

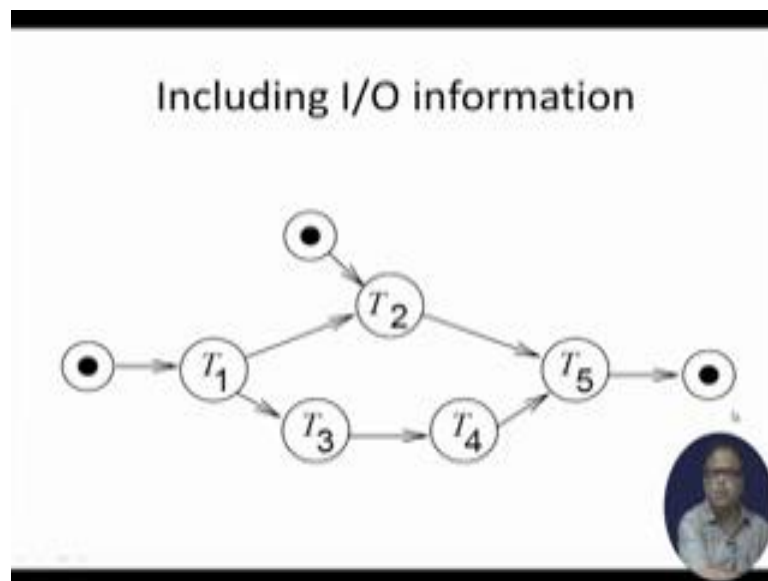


Let us consider an example here. This is a very familiar and simple example of a dependency graph or dependent graph, where we have got five tasks. And, the five tasks dependent in this way; the T 2 is dependent on T 1, T 3 is also dependent on T 1, T 4 is dependent on T 3 and T 5 is dependent on T 2 and T 4. That means, say for example, a T

5 can start only after both T 2 and T 4 complete. T 2 can start only after T 1 completes. So, this is a dependency graph by which you can show the different components. This is the different components, task components. And, there should be some communication about the completion time that this has been completed. It can be either time that this is T 2 is dependent on T 1, the T 2 can start only after T 1 completes. Or, it may be also the case that T 2 can only start when T 1 sense a particular piece of data to T 2. So, there can be different types of dependences, but whatever those dependences are we must respect that while we synthesize.

We can also add some timing information. For example, with respect to T 1, this node, I can say that it can start any time, but must finish within 7 units of time. That is the dead line. T 2 can start at. So, you can see the two brackets. So, it can, it is not inclusive I mean, it is not extendable zero is included. Now, it can start any time after one. At 1 may be, it is arriving at 1 but, must complete by 8 units of time. In that way, I can also add timing constraints to each of these task graphs; timing information I would say.

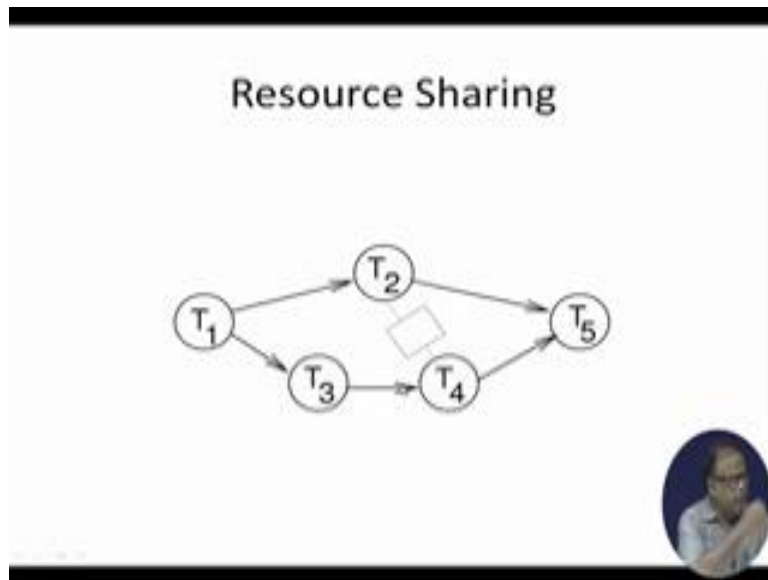
(Refer Slide Time: 06:12)



We can also add I O information as is being shown here. So that T 2 will get. Now, here I am not exactly showing what is the input. We are showing some token. We will come across this notion of the token. We will come across the notion of the token, later on when we discuss petri nets. Here, it just shows that T 2 need some particular input to arrive in order that it can start. So, T 2 needs now two things. One is the completion of T

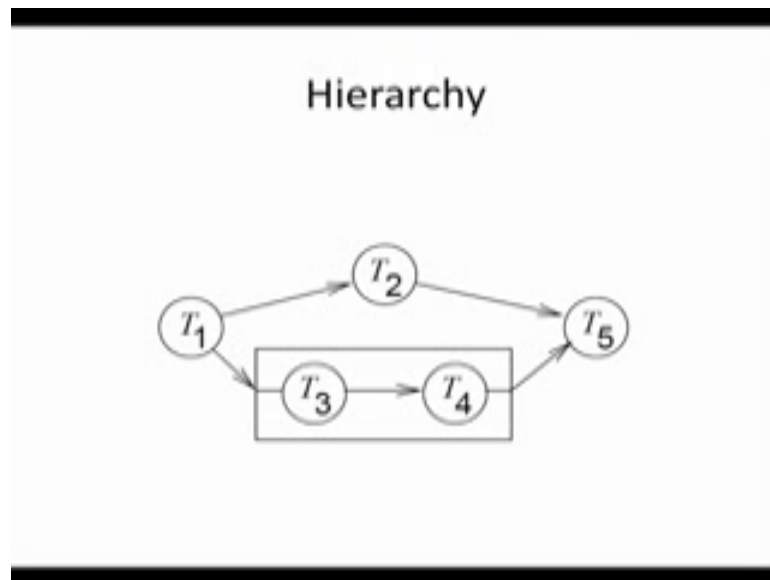
1 or some data to come from T 1. Or, and sorry, and the arrival of this data, input data. So, when both of them are there, T 2 can start. And then, like that here in T 5 when it completes, it will send some output data. So, we can show some I/O information here also.

(Refer Slide Time: 07:13)



We can also designate some resource sharing. Like some particular resource will be share between two tasks T 2 and t four. All this we are doing in order that we can capture the dependence and what are required between what are the constraints in the overall specification or the model of the execution.

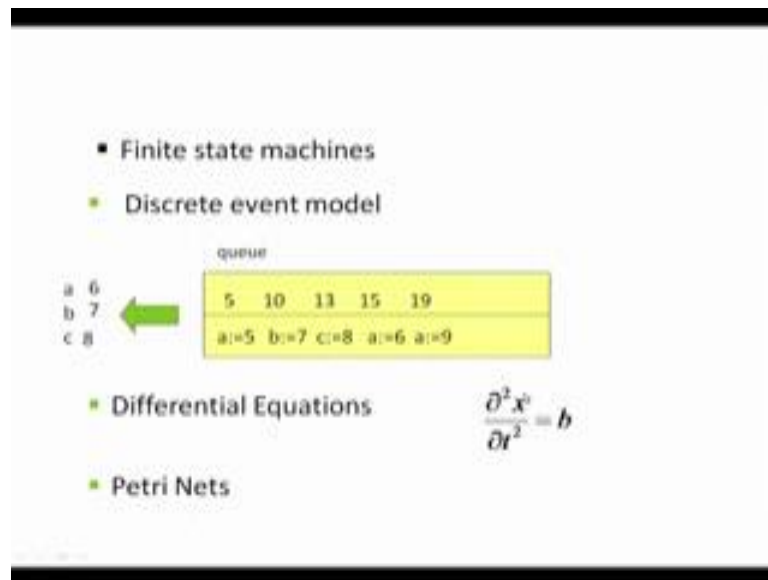
(Refer Slide Time: 07:41)



We have talked about hierarchy earlier. And, here you can see that we can also use hierarchy like I could have instead of T 1, T 2, T 3, T 4, T 5, I could have said T 1, T 2, T 3, T 4, T 5. So, I could have combined this and later on at a little level of hierarchy, little level of decomposition of the specification, I could have said that T 3, T 4, this combined unit, this combined unit is actually T 3 preceding T 4. So, it is not necessary for the complex scenario I have to specify all the details at the top level. So, I can gradually come down.

Now, coming about computing with the components, so we have shown the components and some of the communication among them.

(Refer Slide Time: 08:43)



So, the first is finite set state machine. All of you know what finite state machine is. I will come back to finite state machines again later. But, finite state machines consist of some states. And, one is the initial state, one is the final state. And, there are set of inputs and some outputs. And, there are two functions. One is, one function is mapping from the state transition that given a particular input. I have given a particular state, so I can have a function say f , which maps from a particular state X . X is the state and input to the next state, where X is the set of states. Or, it can be, a function could be that a (Refer Time: 09:58) state transition is going to the new state based on some time.

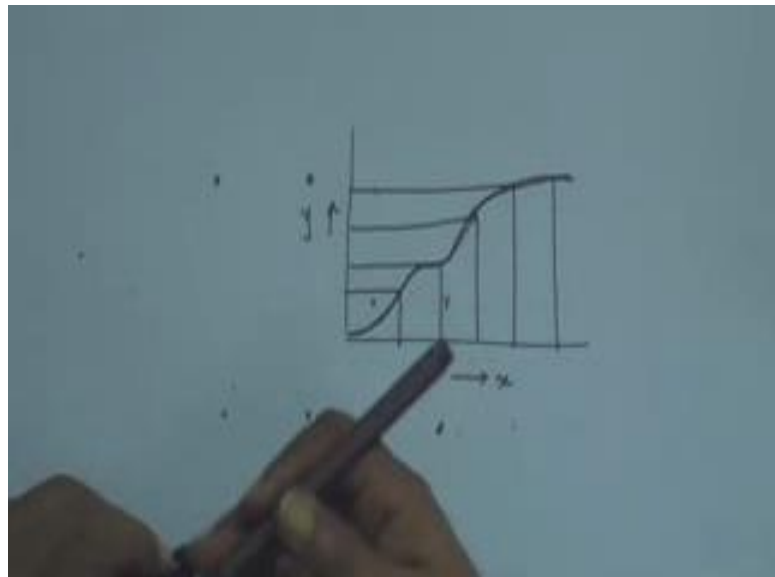
Now, another function that I can have, actually, this is, I will come back to this later. And, there is another function which maps to the X and the input to the particular output; where Y is the set of outputs, possible outputs. So, all of us have seen this finite state machine and will come back to this again.

Now if we look at another scenario that is a discrete event model. Discrete event model, where which is the typical thing that we do in our computer system, in (Refer Time: 11:00) model. Suppose, we have got a queue and these are the timings, now if I, first this part is executed; a is assigned 5. So, a gets 5, next b gets 7, next c gets 8, next again a gets 6, then a gets 9, so on and so forth. So, in that way it proceeds. That is the discrete event system that at every level, that at every interval, discrete interval, some

computation is taking place, and we are having a reflection of that on the memory. And that is what is known as the discrete event system.

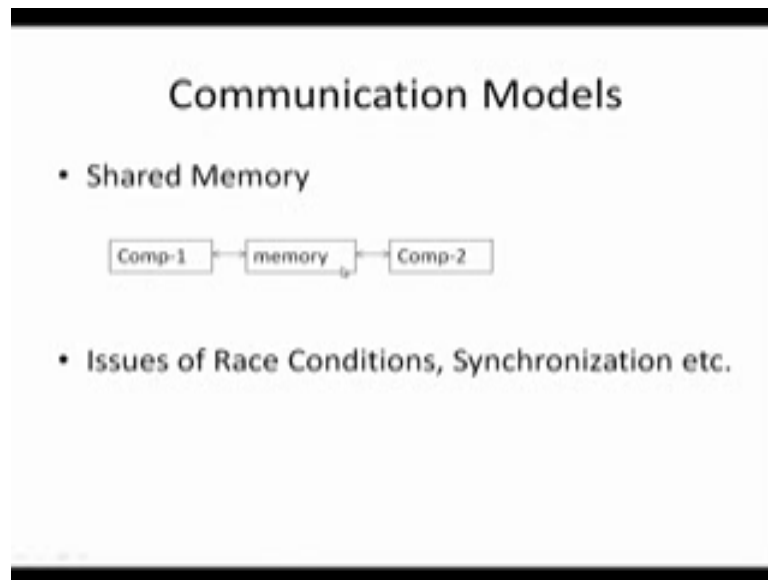
Now, most of the systems that we do and when we program, the program is essential in a one among scenario, a program is a sequential set of, sequential execution of instruction. So at every step, one instruction is being executed. Whenever, we are trying to simulate the real life systems, often we find that some systems are not discrete, they are continuous. For example, the way the temperature changes when I heat a bath of some fluid or liquid. That is a continuous change. So, or say for example, actually when the car accelerates. So, it is the continuous phenomenon.

(Refer Slide Time: 12:43)



But, we often; it is the continuous phenomenon, say it start takes place like this, something like this; continuous where this is y and here is x . but, we actually take some discrete views. We have discussed sampling and all those. And, actually take the data at this distinct point. But, in some applications we need to actually model this continuous behaviour. And for that differential equations like this is the better model. All of you are familiar with differential equation. The other thing is petri nets. And petri nets, we will come across in detail later.

(Refer Slide Time: 13:28)

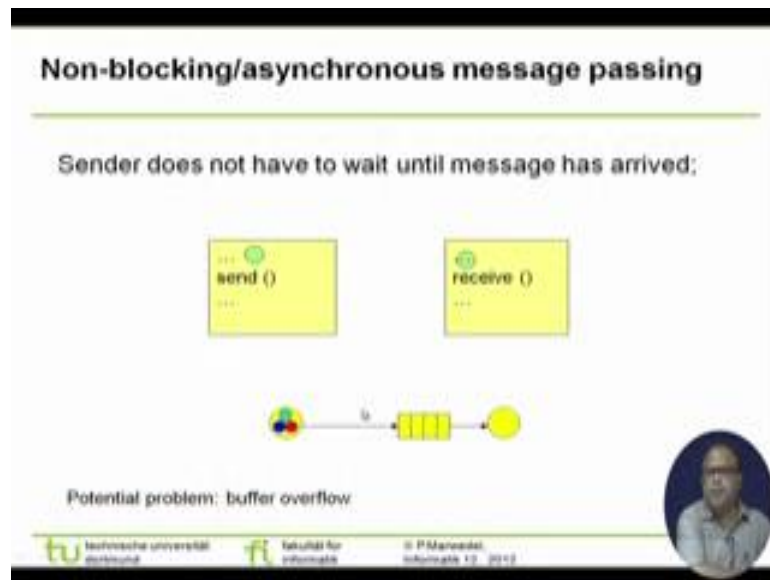


So, we can have different computation models; finite state machines, discrete level, continuous modeling. Nowadays, we are talking of hybrid systems, where we are modeling discrete behaviour as well as continuous behaviour in the same paradigm and, where we are using different representation, so we can capture discrete as well as continuous behaviour. I said that in order to have a complete model, we need the computation. We have to capture the model of computation. Also, we have to capture the communication.

So, the communication; the one of the very common ways which or many of you are familiar with is shared memory communication. There are two processes, two computations here in two different components. And, they are communicating using some shared variable and that shared variable, whenever you call shared variable or global variable, `whatever we say that is the particular shared memory location.

Please, note that this is all again a shared resource that we were discussing earlier. Access to the shared resource, if two computation want to write to the same shared resource, they must be mutually excluded. All those phenomena come into play here. All those issues coming to play here, but shared memory is the nice way of communication. But, there are issues of race conditions, synchronization, etcetera because it is the shared resource. That is one way. We will see each of these things in much more detail.

(Refer Slide Time: 15:18)



The other is message passing. The second; so, one is shared memory, the other one is message passing. Now, what is message passing? The message passing can be synchronous or asynchronous. Synchronous means I am sending messages at particular point of time. Let us see how it looks. Again, I am borrowing it from the slide of the (Refer Time: 15:46). Here, you see asynchronous message passing. That is, I can, at any point of time do it. And, it is a non-blocking. So here is the receiver, here is the sender. And, the sender is not caring so much about what is the status of the receiver. I will simply I am; this green dot is designating the computation. So, when the computation is coming, progressing and coming to this send message part, it will simply send the message here. So, and this was waiting might be.

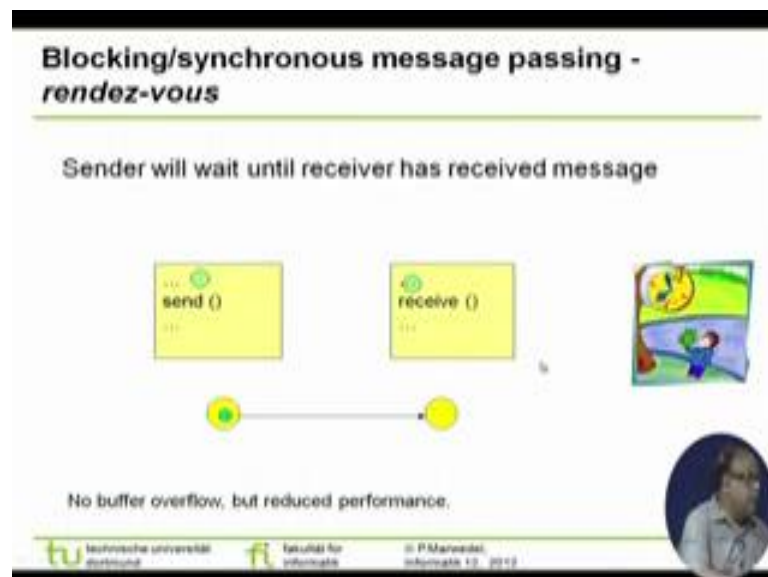
So, what happens is as you see here, this message goes over there. And as soon as it goes over there, it is just like here it is sending it in a queue. I do not really care whether the thing, this one, is consuming at the same rate or not. So, the sender, as the, you can see in the lower figure, the sender is going on sending irrespective of looking at how many places are there in the queue. So, continuously if it goes on sending, this queue can overflow also. So, then it goes to the queue it sets.

(Refer Slide Time: 17:06)



So, the other, here it goes on sending. Consequently, what can happen if the receiver is also fast enough, then it will go on consuming. And, the whole performance of the system will be good because the sender does not have to wait. This is called non-blocking.

(Refer Slide Time: 17:31)



The other way of message passing is blocking or synchronous message passing. For example, here the sender will wait until the receiver has received the message. The receiver will receive a message, till then the sender will wait and will not progress. What

will be the advantage of this? The advantage of this will be that I am sure that no message will be lost. What is the disadvantage of this? The disadvantage is that sometimes if the receiver is not fast enough, the sender will have to wait till the message can be sent.

So see here the sender, this process, comes to the send point, generates the message, sends the message. The receiver receives the message, consumes the message and then tells the sender, sends an acknowledgement that “Yes, I have received it, you can progress”. So, you see here this is coming over there and this receiver is sending an acknowledgement. Consequently, if I come back to the earlier scenario here, this one, if I consider a buffer in between, in the non-blocking mode, if this sender process is faster it will go on sending and the buffer can be full. And, if the buffer is full, then the messages that it is sending will not get a place and will be lost. The sender is not keeping any tag on what is the status of the messages sent.


So, that will not impede the flow of the sender, but we can have the buffer overflow problem. But, in this case we will have no buffer overflow problem, but the performance will be reduced because; may be reduced, I would not say will be reduced, but may be reduced.

(Refer Slide Time: 19:42)

Models of computation considered in this course			
Communication/ local computations	Shared memory	Message passing	
		Synchronous	Asynchronous
Undefined components	Plain text, use cases (Message) sequence charts		
Communicating finite state machines	StateCharts		SDL
Data flow	Scoreboarding + Tomasulo Algorithm (= Comp. Architect.)		Kahn networks, SDF
Petri nets		C/E nets, P/T nets, ...	
Discrete event (DE) model	VHDL*, Verilog*, SystemC*, ...	Only experimental systems, e.g. distributed DE in Ptolemy	
Von Neumann model	C, C++, Java	C, C++, Java with libran	CSP, ADA

* Classification based on implementation with centralized data structures

tu technische universität düsseldorf fl fakultät für informatik © F. Marzetta, informatik 1.0, 2012



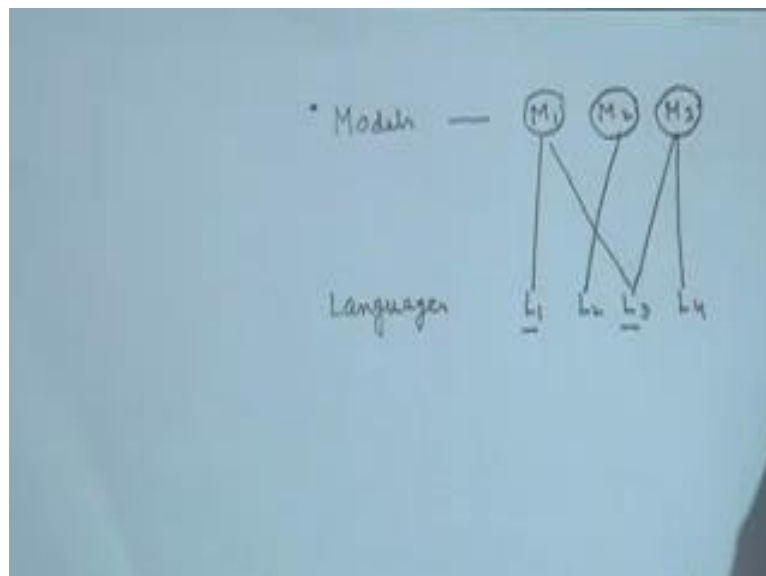
So, we will be considering some models of computation here. May not be all. Plaintext is of course there. That you know. We will talk about state charts, where we are classifying

these based on; we are classifying these based on two things. One is the computation and what is the way of communication. So, the computation models, say communicating finite state machines. First of all, simplest possible is the finite state machine. And, the varieties of them; I mean, there are more sophistication. We will soon see that there are problems with finite state machines. So, you will see state charts. State charts will use shared memory communication. We will see another version of communicating finite state machines, SDL, which is using message passing.

We will try to see something of Kahn networks, which is the data flow. But, certainly will look at petri nets, which is looking at asynchronous message passing. And VHDL, Verilog, you have, you are already familiar with to some extent. So, and C, C ++, Java you are already familiar. So, this part I will not discuss. I will mostly discuss state chart, SDL, Kahn networks and petri nets.

So, next let us move to; move a little further on models and specifications. So, what have we done till now? We have shown what are the requirements or (Refer Time: 21:40) of a modeling language. Now, one is the model there are two things one are I can have different models.

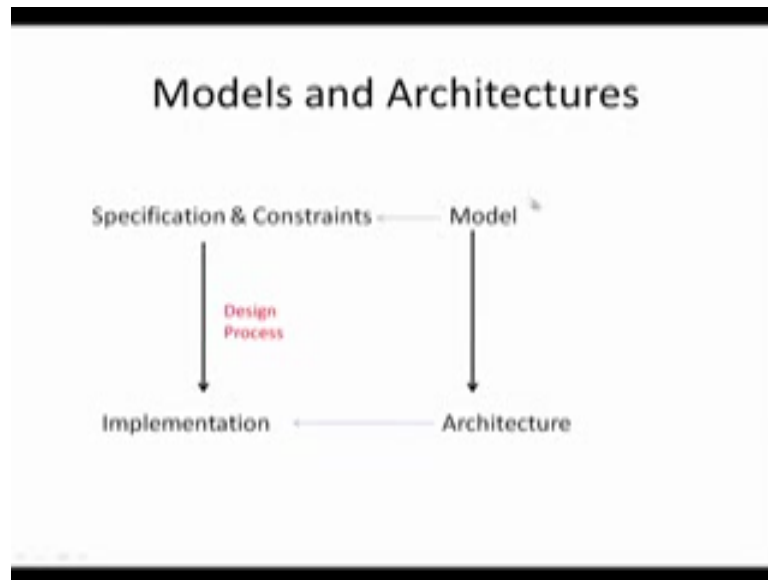
(Refer Slide Time: 22:07)



I can have different models. Again this and, there are languages. Now, it is like a bipartite graph that there can be different models and there can be different languages. And, some models can be captured by some languages; may be L_1 and L_3 is powerful

enough to capture this model M 1, may be L 2 can capture model M 2 and M 3 may be captured by L 4 and L 3 both. So, there are two things. One is the model part and what is the language in which I can represent a model.

(Refer Slide Time: 23:13)



So, we will look at models here; models and architectures. We have seen now. One is we are trying to get some model. You have to first capture the model. This model is actually representing the specifications and we are specifying the constraints.

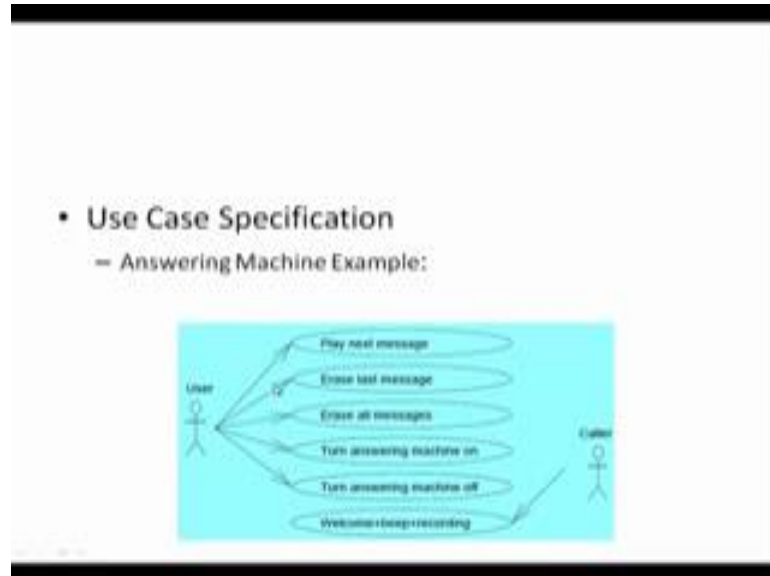
Now, parallelly also we decide on what architecture I want to implement this model. I may like to do it on a GPU, general purpose, GPP, general purpose processor or I can like to put it in on DSP architecture, whatever I wanted to do.

Now, based on this that is another input that I have to take; the design process is actually converting. The design process is actually converting. So, what did I say? I said that when we talk of models and architectures, we start with the model. And that model, from that model we derive the specification and constraints. And, parallelly we need some architecture to be decided. It maybe a multiprocessor unit or it may be a uniprocessor configuration, it may be a DSP, it may be a normal processor or it may be a ASIP. So, that we have to decide.

Now given this specification and constraints, the design process converts the specification and constraint to an implementation. And that implementation takes as

input, what architecture on which you are mapping. We also often call that specification to implementation mapping issue, which we will cover later.

(Refer Slide Time: 25:07)



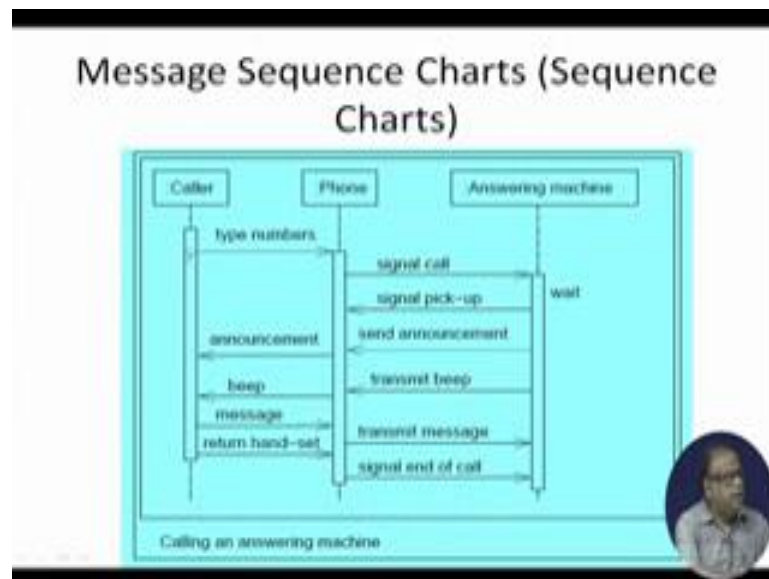
So, given that let us, before we come to this, now we know that the simplest specification is that of natural language but, natural language has got its own ambiguities. So, one simple means of specification was the sequence message, sequence chart, which was later termed just sequence charts, and which finds the place in a very popular system called UML. So where, we want to specify the use case.

Here, I think it is visible. Here, is an example of the use case of an answering machine. Let us see what it says. The caller; on one side there are two users. One is the caller who is calling and here is the person who owns the phone. He is the user. The caller calls. And, for the caller the answering machine will provide what? Some welcome message, then some beep and then the recording facility. So, these are the functions that the caller will get. What are the function that the owner or the user will get? When he comes back home he can play the next message that is there. He can erase the last message, he can erase all messages, he can turn answering machine on, he can turn answering machine off. There should be more. He can record new message. That is not shown here. So, these are the, whenever I am suppose designing an answering machine.

So, this is given to me as a specification that the thing that you design must have the facility of doing all these things. Now, this is again a little too general. It just tells me

these are the things you have to do. It does not tell me whether there is dependency on these are not, whether are all these things independent, are there any timing constraints, which is following what, whether the welcome beep should be given first or what will, which of these will be played before that etcetera. So, we can have much more detailed version. Let us look at these.

(Refer Slide Time: 27:55)

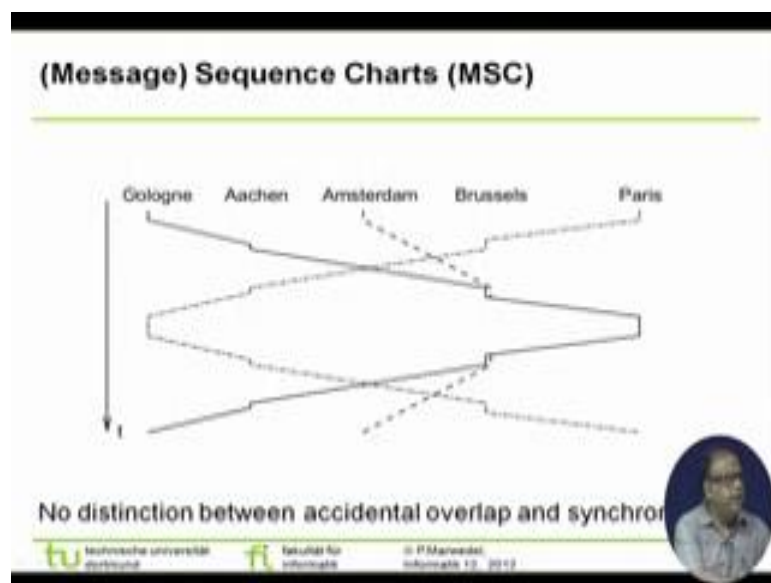


Say for example, the caller. What is? The caller on one side, the user on the other side, the caller is calling an answering machine and here is the phone. Now, this is sequence chart; earlier called message sequence chart. Now, the caller first types the number. The phone gets that and signals the call. The answering machine waits and signal speak up. Some time has elapsed, no one has picked up.

Now while I describe this, the horizontal direction where this mouse pointer is moving is the space and the vertical direction, this, is the time. So, it waits for some time and then the answering machine sends the announcement to the phone. The caller gets the announcement. Just you see the delay; a very little delay. Then, it sends a beep. The answering machine sends the beep. Subsequently, look at this small delay. The phone sends the beep to the caller. The caller sends the message. The message is transmitted to it little delayed to the answering machine and returns the caller, returns the hand set, that means, ends the call. And, the phone is answering. End of call to the answering machine is stopping.

So here we are, but still there are much more details on this. There are much more details on this because might be the suppose the beep has been sent and the message, the user was just remaining silent, not sending any message, then beyond the particular time the answering machine should again send the beep and stop. All those details are not shown here. So, it is a sequential flow that has been shown. There are multiple sequences, conditionalities which is not shown here. But, this is one way. And, there are commercial tools available, using which we can, we can use those tools to specify using message charts or sequence charts.

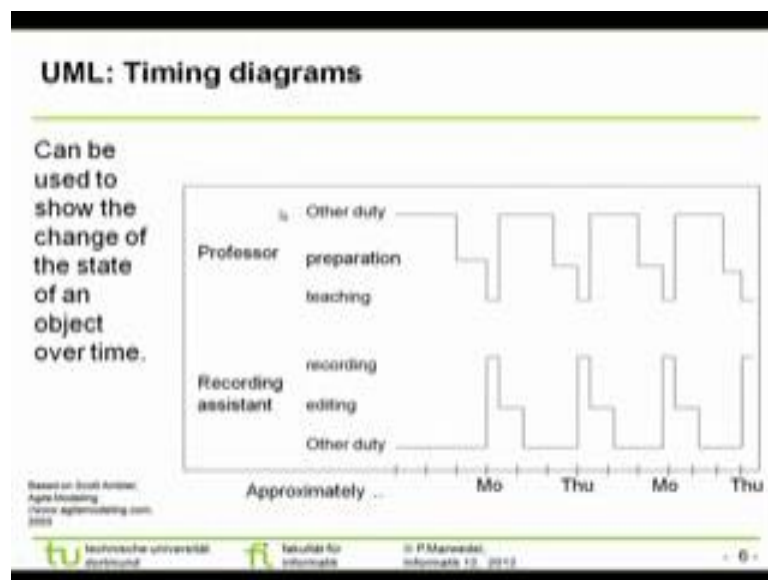
(Refer Slide Time: 30:46)



So, here is another message sequence chart taken from Germany. Here is the time. And, it is showing; you see that there are cities. And, trains are travelling from different cities to another. So, from Cologne, a train starts. It is in Cologne. This space, this space, it is waiting in Cologne for a while. It starts and takes the time sometime to come to Aachen, another city. It waits in Aachen for a while and then travels to Amsterdam. See, the train is quite fast because from what can you understand whether the train is fast or not? From the slope of this curve; if the curve is, I mean this, if it is much more grouping in that case it is slope; the more horizontal it is, the faster it is. So, it is quite fast that it travels from, sorry, it did not travel from Aachen to Amsterdam, it went to Brussels. There, it stopped for a while and went to Paris, waited in Paris for a while, longer time, then travel back to Brussels and (Refer Time: 32:11).

In the meanwhile, another train; there was a shuttle train from Amsterdam to Brussels now. So, here it is coming here. And, there is train from Paris to this. So, we could represent this. But, what is the problem? One problem is that I cannot, suppose if the scenario is this that this train starts from Cologne comes to Brussels and this shuttle train comes from Amsterdam to Brussels and then these two join together. The coaches join together and travel to Paris. Whether that is happening or not? What is happening here that is not shown here very distinct? So, whether these are synchronizing that people or just synchronizing here or this is an accidental overlap that is not very clear in these representations.

(Refer Slide Time: 33:09)



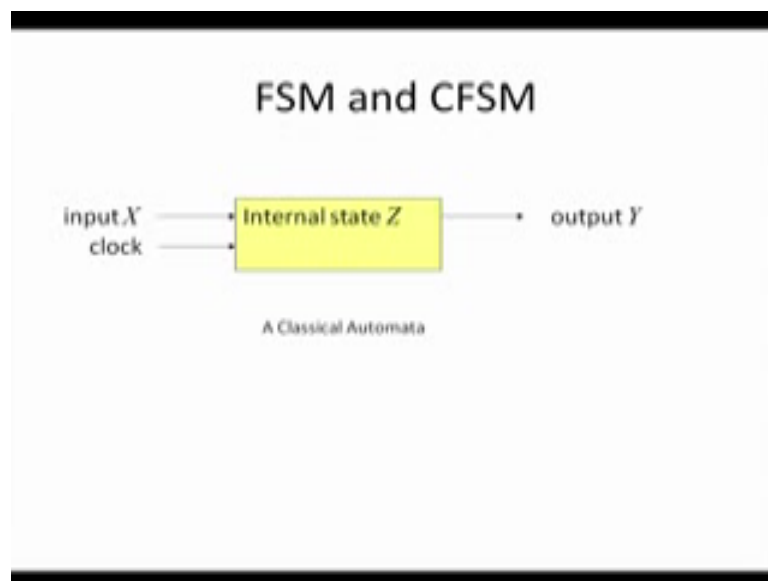
Timing; I will; this lecture, I will conclude with this explanation of this timing. You see the timing here in UML. In UML, where we can specify this using message chart, what are the ways we can specify timing? It is not integrated in the representation. We have to separately show the integration.

Here is the very interesting example, say just as I am recording the lecture, so and there are some recording assistants sitting in the other room were recording. What are their time of engagements? The professor has got many other duties, administrative duties. And, he is doing that. And, all of the sudden now you see this is, this is the time. Now, this is time. So, only suddenly find that well there is the class. So, he has to immediately switch his action, come to the preparation mode, spends preparation for some time and

then comes to the lecture. Lecture takes the least amount of time. So, we provide the lecture and again rush for some other administrative or some other work might be. Lots of works are kept pending. So, those are done. So, this is the timing which a professor is spending. So, much time for the teaching, so much time for preparation. Maybe for this lecture, the preparation time was less than this one.

What is happening to the recording assistant on the other hand? They are also doing other duties. And then, when teaching is coming up, they are immediately leaving their duties and coming to recording. And after recording, when the professor is doing some other tasks, they are busy doing the editing and they finish the editing and then go back to this, other duties. So, in this way we can represent the timing of both that, I mean, more than one processes. And, it is up to the design system to look at this timing and adhere to this timing.

(Refer Slide Time: 35:28)



So, we will start from; we will follow up in the next lecture from this. UML is one method of representation. But, we will now look at finite state machines and other versions, extensions of finite state machine from the next lecture.