Engineering Real-Time Communication Through Time-triggered Subsumption
Towards Flexibility with INCUS and LLFSMs

David Chen, René Hexel and Fawad Riasat Raja
School of Information and Communication Technology, Griffith University, Nathan QLD, Australia

Keywords: Time-triggered Communication, Safety-critical Systems, Software Modelling, Subsumption Architecture, Logic-Labelled Finite State Machines.

Abstract: Engineering real-time communication protocols is a complex task, particularly in the safety-critical domain. Current protocols exhibit a strong tradeoff between flexibility and the ability to detect and handle faults in a deterministic way. Model-driven engineering promises a high level design of verifiable and directly runnable implementations. Arrangements of logic-labelled finite-state machines (LLFSMs) allow the implementation of complex system behaviours at a high level through a subsumption architecture with clear execution semantics. Here, we show that the ability of LLFSMs to handle elaborate hierarchical module interactions can be utilised towards the implementation of testable, safety-critical real-time communication protocols. We present an efficient implementation and evaluation of INCUS, a time-triggered protocol for safety-critical real-time communication that transcends the rigidity imposed by existing real-time communication systems through the use of a high-level subsumption architecture.

1 INTRODUCTION

Nowadays, a model-driven software development approach is widely used by developers in contrast to lower level implementation approaches, as it assists in developing, faster and simpler modules and applications (Estivill-Castro and Hexel, 2013b). Finite State Machines (FSMs) or behaviour trees are used to represent high level specifications of behaviours. This kind of modelling approach fulfills the agenda of Model Driven Engineering (Schmidt, 2006) for software development. In contrast to other, more traditional implementation approaches, Logic-Labelled Finite-State Machines (LLFSMs) (Estivill-Castro and Hexel, 2014) allow translating requirements into high-level, executable models (Billington et al., 2011a). These are less susceptible to implementation errors as models can be directly interpreted, simulated, verified, and executed on a large number of platforms, including embedded control systems (Estivill-Castro et al., 2012).

In control systems, where different modules are interacting with each other, it becomes very important to predict the results and shield the details of one module from others. To solve this problem, the subsumption architecture (Brooks et al., 1986) has proposed behaviour based decomposition of such complex systems into layers of increasing level of abstraction, where high level layers can subsume the lower level layers. Several other similar approaches (Kaelbling, 1987; Payton, 1986; Arkin, 1987) were developed but one big advantage of the subsumption architecture is the ability to cater for the evolution of the complexity of a control system by accretion of higher-level layers. This approach allows the incremental development of a control system, as addition of each new layer provides a new additional behaviour to the controller. Further layers can be added on top of the existing layers without affecting their behaviours. This way, we can always have a functional controller with each new behaviour throughout the development process. So far, the subsumption architecture has largely been used to build robotic control systems (Connell, 1987; Brooks et al., 1988; Mataric, 1990; Brooks, 1987).

In this paper, we use LLFSMs as a modelling tool, where transitions from one state to another state are based on expressions in logic rather than events. This not only reduces the overhead significantly as, for example, no memory allocations are required for event queues, but also makes system performance more predictable. Although modelling with LLFSMs is a very...
effective approach as shown in the literature (Billing-
ton et al., 2011b; Estivill-Castro and Hexel, 2013a),
to our knowlege, no attempts have been made to date
to use them towards the implementation of a safety-
critical, hard real-time system. We not only discuss
due to the implementation of INCUS (Chen et al., 2014)
using LLFSMs, but also show how the subsumption
architecture helps prevent design issues and how
an arrangement of LLFSMs has proven a better tech-
nique that enables to design and develop a more com-
plex protocol faster.

2 REAL-TIME
COMMUNICATION FOR
SAFETY-CRITICAL SYSTEMS

INCUS (Chen et al., 2014) is a communication proto-
col for distributed safety-critical real-time systems. A
real-time system distinguishes itself from other sys-
tems in that it is required to provide services within
defined time frames, which means it must meet cer-
tain deadlines. A deadline is the time limit by which
a task must be completed. Results delivered after
due to their deadlines lose some (in the case of soft dead-
lines) or all (in the case of firm deadlines) of their util-
ity. Safety-critical systems are systems where a fault
or missing a deadline can have severe consequences
such as injury or death, i.e., they are hard real-time
systems where missing a firm deadline can poten-
tially have catastrophic outcomes (Kopetz, 2011). To
prevent faults, safety-critical systems typically use
means of redundancy, such as replication and distri-
bution to tolerate or recover from these faults.

In distributed safety-critical real-time systems,
different nodes are connected with each other through
a shared communication channel such as bus. They
coordinate their actions through message passing,
therefore, timely and reliable message delivery is crit-
ical. Communication errors and unpredictable delays in
transmission may lead to unpredictable behaviour of
distributed real-time systems. For instance, con-
sider a brake-by-wire system in a car. When the driver
hits the brake pedal, then, based on physical param-
eters such as the speed of the car and its wheels, a brake
force is calculated and transmitted to each wheel for
stopping the car. A slight delay or error in com-
munication may lead to a longer stopping distance or
even complete brake failure, potentially causing
harm. When nodes are sharing a single communica-
tion channel, generally one node at a time, is allowed
to transmit over this channel. Otherwise, simultane-
ous transmission by multiple nodes would interfere
with each other. This is known as a transmission col-
lision and all the nodes involved in the transmission
can lose their messages. Typical recovery techniques
such as positive acknowledgement and retransmission
(PAR) significantly differ in their performance be-
tween the best and the worst case and therefore ap-
proaches that do not impose different timing in the
case of faults (Lamport, 1984).

In light of these issues, the Time-Triggered Ar-
chitecture (TTA) (Scheidler et al., 1997; Kopetz and
Bauer, 2003) was introduced, which uses a Time Di-
vision Multiple Access (TDMA) scheme for chan-
el access. Communication time slots used by the
nodes that make up the distributed system are de-
terministic and scheduled offline (Scheidler et al.,
1997). Therefore, all nodes know exactly at what
point of time each node will transmit its message
and this guarantees that channel access will be free
of collision. Well known communication protocols
for safety-critical real-time systems such as the Time-
Triggered Protocol (TTP) (Kopetz and Grünsteidl,
1994) and FlexRay (Berwanger, 2001) use this ap-
proach for safety-critical real-time communication.
To this end, they use a scheme where nodes are al-
located static and equal length time slots to transmit
their information in each TDMA round of a cluster
cycle. However, this static scheme comes at the ex-
pense of flexibility. Unfortunately, in traditional ap-
proaches towards designing real-time communication
protocols, this tradeoff cannot be easily avoided and
so far, approaches that attempted to introduce a sig-
nificant amount of flexibility (Andersson et al., 2005;
Li et al., 2009) have failed to offer the same levels of
dependability or only offer flexibility for non-safety-
critical messages (Berwanger, 2001). We now show
how this tradeoff can be minimised in INCUS.

3 INCUS

Before discussing the implementation details of the
communication protocol using LLFSMs, we will
briefly discuss here the principles of operation of IN-
CUS.\(^1\) INCUS is building on previous TTA proto-
cols (Kopetz and Grünsteidl, 1994; Berwanger, 2001)
and is using a TDMA approach for channel access,
but unlike other TTA communication protocols, it
supports variable length node slots based on the trans-
mision payload of each node in each TDMA round
of a cluster cycle (Chen et al., 2014). In contrast
to protocols such as TTP and FlexRay, this offers
a higher degree of flexibility for each node to only

\(^1\)For reasons of space, the full specification of INCUS is
not replicated here, but can be found in Chen et al. (2014).
transmit as much information as necessary, but increases the complexity of the implementation. The length of each message slot is configured in a data structure called Message Descriptor List (MEDL). To support deterministic communication, each INCUS node has a replicated copy of the MEDL. The MEDL not only stores the length of the individual time slot, but also holds the time schedule for all nodes, i.e., the exact time within the TDMA cycle when a node can transmit and receive data to and from other nodes is defined in advance. INCUS uses three types of frames for communication: Normal frames (N-frames), Initialisation frames (I-frames) and Coldstart frames (CS-frames). N-frames carry application data, I-frames carry information for reintegration of lost/recovering nodes and CS-frames carry information for integration of all nodes during system start-up.

As INCUS is designed to be used as a communication protocol for safety-critical real-time systems, it provides fault tolerance in order to stay operational in the presence of faults and maintain the reliability and safety of the protocol (Chen et al., 2014). A key criterion for fault tolerance, clique avoidance, and the ability to form the basis of a fail-operational system, the protocol uses a membership service in following the principles of the TTA (Kopetz and Bauer, 2003) to keep track of active and inactive nodes after each TDMA round. Correspondingly, each node stores its view of the fault-free operation of other nodes (as perceived when receiving messages from these nodes) in its membership vector. This membership service also acts as an acknowledgement scheme without the overhead of explicit membership transmission during normal operation or the requirement for acknowledgments from receiver nodes. Implicit transmission of the membership vector of each node is done through frame CRC calculation, by incorporating the membership vector (and other vital controller state information such as the current MEDL position) in the CRC calculation of the sending and receiving nodes, without actually transmitting this information. State agreement therefore is confirmed through a CRC match, and nodes that have a different point of view from the sender will detect a CRC error and remove the sender from the membership vector. Clique avoidance (Kopetz and Grünsteidl, 1994) ensures that these minority nodes will no longer receive frames from majority nodes until they restart and re-integrate into the cluster again. Minority nodes are able to detect that they are in the minority through a membership vector that indicates that no more than half the nodes are active. In other words, if fewer than half of the nodes agree with a node, that node knows that it no longer is part of a majority, causing it to restart and re-integrate into the system based on I-frames transmitted by other nodes.

Clock synchronisation is a core requirement of the TTA and thus, INCUS uses a robust clock synchronisation mechanism following the principles established with TTP (TTEch, 2004). To maintain time, all the nodes have physical clocks and a predefined time schedule. Each node operating in the system must synchronise its transmit or receive actions according to the predefined time slots. This is only possible if the nodes’ clocks are synchronised. A difference between the expected arrival time (defined in the MEDL) and the actual arrival time of a frame at receiver node is measured and then used to compensate for the clock skew between the clocks of sender and receiver nodes. The Fault Tolerant Average (FTA) algorithm (Kopetz and Ochsenreiter, 1987) is then used to correct the clock of each node based on the measured deviations.

4 EXECUTABLE COMMUNICATION MODEL

When implementing a communication protocol, a key design decision is the choice of tools and the level of modelling for this implementation. The prototype software for TTP/C, for example, was designed at a low, procedural level and implemented using a mix of C++ and assembly language (Kopetz et al., 1997). While this approach certainly offers predictable, high performance (short only, perhaps, of a direct hardware implementation), a key disadvantage is the design and development effort (several man years) required by such an approach. Moreover, low-level software is often wedded to a specific hardware and difficult to port to a different platform. Nevertheless, to date, the rigorous timing requirements that need to be modelled early on in the design process has made it difficult to model verifiable executable real-time behaviour at a high level (Estivill-Castro and Hixel, 2015). These and other difficulties often encountered with high level engineering of software has often prompted the question of whether it makes sense to engineer software, and hence, there is now a trend to view engineering as craft supported by theory, leading to best practices in software engineering (Jacobson and Seidewitz, 2014). A common element in software architectures in general, but particularly in control software, are finite-state machines. In fact, the most commonly used artefact for the description of software behaviour in UML are state diagrams (Erickson and Siau, 2007; Reg-
Logic-labelled finite-state machines (LLFSMs) are Turing complete, making them fundamentally equivalent to any mechanism to model system behaviour, with key advantages (Estivill-Castro and Hexel, 2015) that shall make them the preferred model here. First, they offer a very clear semantics of concurrency, tremendously simplifying the cognitive burden for the developer (Estivill-Castro and Hexel, 2015). Importantly, though, their execution semantics much more closely resembles the principles of the TTA (Kopetz and Bauer, 2003) and is in direct contrast with the optimistic best-effort approach of event-driven systems (Estivill-Castro and Hexel, 2015). We implemented INCUS through executable models using LLFSMs, and to our knowledge, this is the first attempt at implementing software suitable for safety-critical hard real-time systems using this approach.

### 4.1 LLFSM Design of INCUS

In our first iteration, we embedded all the lower level and higher level implementation details in a single LLFSM. To this end, we modelled INCUS in stages, starting from the very basic functionality of the protocol, i.e., transmit and receive message at a predefined point of time, following the time-triggered approach (Chen et al., 2014) of the specification. Then, we added the basic fault tolerance algorithms and mechanisms (FTAMs), e.g., the Cyclic Redundancy Check (CRC) whether a received message is corrupted. Step by step, we incrementally added additional behaviours required by the protocol to take it towards the full specification. Each of the steps was designed using MiCASE (Estivill-Castro et al., 2014) and compiled to an executable that was run, tested, and verified using clfsm (Estivill-Castro et al., 2014). What is important to note is that, since LLFSMs represent executable models, each of these steps, despite not yet implementing the full specification, gave us a fully functioning, executable prototype that we were able to simulate, execute, test, and validate.

The complete INCUS specification is described in Chen et al. (2014) but here, we will focus on one key aspect and briefly discuss the implementation of the transmission behaviour of INCUS using a single LLFSM as shown in Figure 1. In the Initial state, all the necessary parameters are initialised such as the Message Descriptor List (MEDL) that holds the time schedule for the data transmission and reception phase for all nodes. Each node starts from slot zero as their first slot in the Set Slot state, then transitioning straight to the Slot Pos state. In this slot, each node will check the MEDL to figure out whether it needs to act as a sender node or receiver node according to the current slot position. Note that, since we are discussing the message transmission mechanism of INCUS, we are ignoring the receiver part (as a receiver node) of INCUS in this example, but the receiver follows analogous steps to the transmitter states discussed below. So if in the current slot, the node is meant to act as a sender node, a corresponding transition is made from the Slot Pos state to the CState S state. The Controller State (C-State) is initialised in this state so that CRC value can be calculated over the C-State. This allows a message to be rejected as...
incorrect, not only if there is a physical transmission error, but also if there is any other fault that causes C-State disagreement (Kopetz et al., 1997).

After initialisation of the C-State, the node transitions to the next state CRC_CState_S. From there, it takes a byte at a time and calculates the CRC on each individual bit of that byte and when finished, takes the next byte. This continues until there are no more data left for transmission. This whole procedure is achieved through the state transitions from the CRC_CState_S, CRC_CState_Next_Bit_S, and CRC_CState_Next_Bit_S states. The next state after CRC calculation is the Wait_Send state, from which a transition is made to the Send_Data state, only once the time at sender node is equal to the time defined in the MEDL for the actual message transmission. This time is termed the slot start time and implements the essential time trigger for the sender node.

It is important to note that, other than the conditional transitions shown in the figure, there is no conditional code here, making, together with the deterministic scheduling of clfsms (Estivill-Castro and Hixel, 2015), the execution time of the compiled code extremely predictable in correspondence with the structure of the high-level model. In fact, the worst-case execution time (WCET) is essentially the same as the best-case execution time, minimising the temporal jitter of the transmission start state. In this state, the Send_Data state, the original message and number of bytes of the message are fetched from the MEDL and then the next state is the SendBYTE state. This state takes one byte of the message at a time, and transmits it bit by bit (through the Clock_Low_S, Write_Bit, and Clock_High_S states, also updating the CRC at each step), and then looks for the next byte (Next_BYTE_S).

Once no bytes are left to transmit (nd->bytes == 0), the sender transitions to the Send_CRC state, where all the bytes of the CRC are transmitted. The mechanism of CRC transmission is same as the transmission of the original message and the states used for transmitting CRC value (CRC_CLK_Low_S, Write_CRC_Bit, Clock_High_S_CRC, and Next_Bit_S_CRC), have analogous functionality to the data transmission states above, but transmit the CRC instead. The Finish_S state concludes the cycle, incrementing the MEDL slot position and transitioning straight back to Slot_Pos if there are more MEDLs slots to operate on, or back to Initial, if the end of the TDMA cycle has been reached and the above steps repeat from the beginning of the MEDL (slot zero).

One pattern that becomes apparent in this initial design is a replication of concerns. In other words, despite the fact that the above description only details the transmission phase of the protocol, we already have replicated the relevant states used in message transmission, i.e., the states required for transmitting the CRC essentially mirror the states used for transmitting the message payload. The states representing the receiver very much mirror the sequence described above, with only minor differences, such as the provision of a small receive window to compensate for clock drift and jitter. Up to this point, the complete LLFSM, including the receiver, already contains fifty states and we have not yet implemented important parts of the communication protocol specification, such as the behaviour for the node start-up and reintegration, clock synchronisation, the membership service, mode changes, and other FTAMs such as the detection of transmit and receive errors on the basis of different timeout parameters. As this single LLFSM grows bigger, it becomes more complex and was nearly impossible to add remaining behaviours of the protocol by adding and replicating more states.

Nevertheless, this initial implementation already serves as a very important proof that it is not only possible to implement a protocol for safety-critical real-time systems using LLFSMs, but also that it can be done much more rapidly (several weeks vs. a few man-years) at a high level, yet yielding fully executable models at every stage. This leads us to the next stage of considering a refined approach that greatly enhances the modularity of the design.

5 INCUS SUBSUMPTION

To reduce the complexity of the overall design and increase the modularity, we follow the principles of the subsumption architecture (Brooks et al., 1986), allowing us to split out functionality into modules that can hierarchically be subsumed by higher level modules. Arrangements of LLFSMs allow the implementation of a subsumption architecture by integrating a number of different finite-state machines, each forming a component or module that can be deactivated using a suspend operation or activated using a resume or restart operation\(^2\) (Estivill-Castro and Hixel, 2013a).

State machine vectors formed by an arrangement of LLFSMs make the decomposition of sub-behaviours into modules particularly straightforward. We already identified repetitive the sub-behaviours, such as the transmission of CRC vs. payload data, and

\(^2\)With LLFSMs, the restart operation simply restarts the machine from its Initial state, while the resume operation resumes from the previously active state.
splitting these elements out into individual modules is as simple as factoring out those states into an individual sub-machine. A decomposition of our earlier, single LLFSM implementation of INCUS into the following four sub-LLFSMs is shown in Figure 2:

1. INCUS\text{-}MAIN\text{-}LLFSM
2. INCUS\text{-}CRC\text{-}LLFSM
3. INCUS\text{-}SENDER\text{-}LLFSM
4. INCUS\text{-}RECEIVER\text{-}LLFSM

The INCUS-MAIN-LLFSM module acts as a high-level master-LLFSM that actually controls the behaviour of the other sub-machines. These sub-machines are composed of the CRC-LLFSM, SENDER-LLFSM, and RECEIVER-LLFSM machines. The Main LLFSM runs concurrently with the sub-LLFSMs and only has the principle purpose of implementing the high-level stages of the protocol and to suspend and restart the sub-machines. These sub-machines are the modules in the subsumption architecture that implement the corresponding underlying behaviours, when required.

In the previous section 4.1, we took the example of message transmission in INCUS using a single LLFSM to highlight the issue of design complexity. In the following section, we will demonstrate how we tackle this complexity issue by implementing the message transmission behaviour of INCUS using the subsumption architecture.

5.1 Tackling Design Complexity using Subsumption

The implementation of the transmission behaviour of INCUS using the subsumption architecture is done by decomposing the single LLFSM from Figure 1 into the three sub-LLFSMs 1–4, i.e., INCUS\text{-}MAIN\text{-}LLFSM, INCUS\text{-}CRC\text{-}LLFSM, INCUS\text{-}SENDER\text{-}LLFSM, and INCUS\text{-}RECEIVER\text{-}LLFSM. Figure 3 shows the main machine. The main machine acts as a master LLFSM and can run concurrently with the sub-LLFSMs. While the subsumption architecture allows multiple sub-machines to operate concurrently, and while the execution semantics of LLFSMs is clearly defined to avoid concurrency issues or temporal inconsistencies (Estivill-Castro and Hexel, 2015), we deliberately kept the design of the INCUS implementation simple, not requiring the concurrent operation of multiple sub-machines at the same time. This greatly simplifies WCET measurement and further reduces the design and validation complexity of the system.

The overall transmission mechanism follows the steps discussed in Section 4.1. After initialising the relevant protocol parameters in its Initial state and verifying, in the Chk\_Slot\_Pos state, whether the current node is the sender node. If the node is the current transmitter, it transitions to the new state CRC\_CState\_S. To perform the CRC calculations over the local C-State prefixing the payload
transmitted in the message, the main LLFSM will now activate the CRC module (Figure 4) by using restart at (machine_id+NODE_CRC).

The CRC LLFSM runs concurrently with the main machine and performs the CRC calculation over the data referenced by the main LLFSM. In the case of the CRC_MACHINE_S state of the main machine, the CRC data reference points to the C-State that the CRC shall be calculated over. The states of the CRC LLFSM are same the ones described in Section 4.1, but the clear advantage here is that we do not need to replicate these states multiple times, whenever we are required to perform a CRC calculation. In fact, this calculation is irrespective of the node’s current role as a sender or receiver, and thus, unlike in the previous implementation, no further replication is necessary.

While the main machine technically runs concurrently with the CRC module, activating the CRC module does not require any concurrent operation, so the main LLFSM simply transitions to the Wait_CRC_CState_S_F state (through the transition labelled is_running_at (machine_id+NODE_CRC)) where it waits for sub-machine completion through use of the is_suspended_at (machine_id+NODE_CRC) predicate. To notify completion through this predicate, the CRC LLFSM will simply suspend itself by using suspend_self() in its CRC.Done state after having completed calculating the CRC value. This is semantically equivalent to subsumption akin to the UML sub-machine notation (Estivill-Castro and Hexit, 2013a).

As soon as the CRC calculation has concluded, the main machine transitions to the Wait.to.Send state, where it waits for the arrival of the transmission slot action time. To transmit the message, the main LLFSM now activates the sender LLFSM (Figure 5), while again simply waiting for completion by sitting idle in the Wait.Send.Data_F state until the sender LLFSM has completed and suspended itself. Unlike the example from Section 4.1, where the transmission logic had to be replicated for transmitting the message SRC, the same sender LLFSM can now be used and will be restarted by the main machine when in order to send the CRC value as implemented by the Send_CRC and Wait_Send_CRC_F states in the main LLFSM. So contrary to the implementation of transmission behaviour using the single LLFSM, the subsumption architecture eliminates the complexity imposed by state-replication, while maintaining the abil-
ity to implement real-time behaviour following the same, conceptual design principles.

5.2 Adding New Behaviours using the Subsumption Architecture

We will now briefly describe the subsequent, iterative steps that were conducted using the subsumption approach. It would have been hard to continue the modelling of INCUS using a single LLFSM due to the complexity explosion alluded to earlier. In the following analysis we will show how straightforward it now is to add a new behaviour while retaining all the functionality of the existing behaviours of INCUS using the subsumption architecture and arrangements of LLFSMs.

5.2.1 Start-up and Re-integration of Nodes

So far, we assumed in our implementation that all nodes successfully resolved their start-up collision scenario (TTTech, 2004) and they are in the state where they have synchronised clocks and ready to transmit/receive messages. To implement system startup and reintegration in accordance with the INCUS specification (Chen et al., 2014), we need to add another sub-LLFSM as shown in Figure 6. We named this machine RE_INTEGRATION LLFSM in a sense that this LLFSM is, in the fault-free case, used only once to run the start-up scenario when all the nodes are turned-on initially. After this, the main LLFSM is used most of the time, but has the ability to trigger a restart to re-integrate a lost node to the cluster of nodes in case of a fault that requires re-integration. The re-integration LLFSM acts as a master-LLFSM only when all nodes are turned-on the first time. It will suspend all other sub-LLFSMs and runs the node start-up algorithm. Once all the nodes are up, the RE_INTEGRATION-LLFSM will suspend itself after starting the main LLFSM. Now the sphere of control shifts to the main machine as above, which will perform the normal operation of the protocol as described. Importantly, a comparison between Figure 2 and Figure 6 shows that addition of this new behaviour has been achieved by just adding a single new layer on top of INCUS_MAIN_LLFSM. Most importantly, the main machine did not require any modification or change to the structure of the previously existing layers of LLFSMs.

6 CONCLUSION AND FUTURE WORK

In summary, we have shown that a high-level implementation of a communication protocol for safety-critical real-time systems based on the subsumption architecture is not only possible, but facilitates the incremental development of the system using executable models throughout. We have shown that with an INCUS implementation based on the arrangement of LLFSMs, the scope of the subsumption architecture is not limited to modelling the behaviours of traditional control systems, but we can also use it to develop finite-state machines with predictable execution semantics and timing. We have demonstrated that LLFSMs support system development of INCUS in an iterative way or in stages, where we can execute, test and refine a safety-critical real-time system at a given level before starting a new level. This allowed us to ultimately implement a more flexible communication protocol in comparison with existing TTA based communication protocols, where the implementation, refinement, and validation was a lot more complex. Our modelling technique has been shown to make feasible designing flexibility into communication protocols with the strict predictability and timing required by dependable real-time systems. Furthermore, we have shown that the complexity of state-replication can be avoided very effectively by using the subsumption architecture provided by arrangements of LLFSMs when developing a communication system, without losing the fundamental properties of predictable real-time performance. The subsumption architecture made it possible to incrementally refine our implementation by adding, modifying, or changing the behaviour of a sub-system without interfering with unaffected components of the system.

So far, we modelled and implemented a safety-critical cluster of INCUS using LLFSMs. In future, we are aiming to design a more flexible and rather more complex model of INCUS where more than one safety-critical clusters will be linked through a network that also carries a non real-time traffic. We expect that this will allow us, for example, to connect safety-critical sub-systems over the internet or other unreliable networks. We intend to adopt the same modelling approach discusses in this paper and we will evaluate the effectiveness of this approach i.e. subsumption through LLFSMs for designing this hybrid and more complex real-time communication. In this case, the mechanisms of the protocol regarding timing requirements and fault-tolerance for safety-critical traffic will be highly complex, but based on the results presented here, we expect this approach to
demonstrate that even highly-complex systems can be implemented using our technique while maintaining system dependability.

REFERENCES


189, Barcelona, Spain. SCITEPRESS Science and Technology Publications.


