

Digging into Concurrency

Angie Chandler, Serena Patching, Lynne Blair

*Computing Department
Lancaster University, UK*

email {angie or lb @comp.lancs.ac.uk, serena.patching@virgin.net}

Abstract

The topic of this paper is the design and implementation of an interacting Lego digger and dumper truck through the use of Petri nets. The focus of this is primarily on the use of Petri nets in the developing of dependable systems, an area of particular concern in the full-size equivalent of our experiment. The content of this paper will discuss the progression of the Petri net model of these two Lego robots, from design to implementation, and finally evaluation. The paper also features an extension to the Petri net, intended to allow the dumper to authenticate the digger for added security.

1. Introduction

This paper discusses the design and implementation of a model representing a digger and dumper truck interacting on a building site with the use of formal, concurrent modelling methods. The idea owes a degree of its conception to the previous work completed on an autonomous excavator at Lancaster University, as discussed briefly in section 2, but here focuses more on the interaction between a digger and a dumper truck, rather than solely on the details of the digger's execution. To this end, a model digger and dumper truck were constructed with the use of the Lego Mindstorms [10] [1] kit in order to emulate the sense of the excavator itself, and an accompanying dumper truck, without the difficulties associated with larger scale models.

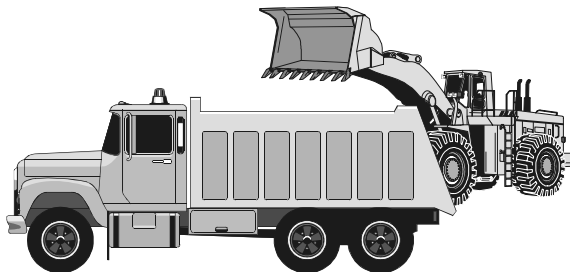


Figure 1 Digger and Dumper Truck - Concept

The primary modelling method discussed in this paper is the Petri net. Here, the Petri net is used to describe the tasks which the digger and dumper truck must perform, and later used to generate software directly equivalent to the Petri net model, providing a great deal of assurance about the software being used.

Petri nets are generic enough to provide the capacity for application to a wide variety of applications, although due to their asynchronous nature they are more commonly used for distributed systems [3] and other similar processes. However, their uses in distributed systems by no means exclude applications to the field of robotics. In fact, robots can themselves form part of a distributed system, as can be seen through the example of an orange-picking robot with point-to-point communications [5], and even the application demonstrated here. Other areas of robotics can also find use for Petri net modelling as a method of eliminating deadlock and other temporal inconsistencies [19] [4], although these properties require testing through timed Petri nets, an extension which has yet to be made to the analysis system used here. There are also examples of use of Petri nets to facilitate co-operation between multiple robots [20], or between a human and a robot [12]. The range of applications for Petri nets is enormously diverse, and limited only by the range of tools available to implement these possibilities.

Our use of Petri nets as a modelling tool for mobile robots, was initially inspired by their ability to represent both the data flowing in the system and the state of the system, simultaneously, in addition to their capacity to represent concurrently executing program threads. This initial interest was then furthered by the ease with which the model could be translated into executable code, as required by the TRAMP toolkit discussed briefly in section 4, without the need to alter any of the components modelled.

The Petri net is put into use here as both a testable model and an eventual implementation device, but an extension to this is also discussed towards the end of the paper, revealing an optional Petri net plug-in which will be used to provide authentication for the dumper truck so that no foreign diggers are making use of it. This

authentication procedure makes use of Kerberos [8], and is discussed in section 6.

2. Background

In recent years there have been a number of approaches taken towards the autonomous execution of the JCB801 mini excavator LUCIE (Lancaster University Computerised Intelligent Excavator – Figure 2) [2]. These have included research into the autonomous use of the digging arm, and also the navigation of the excavator using a GPS receiver [18][6]. These projects have also investigated various approaches to the safety of the excavator [15], particularly with the use of a safety manager as an overseer to the rest of the LUCIE software, which operates using three separate PC104 computers.



Figure 2 LUCIE

The approach discussed in this paper highlights an alternative approach, potentially complementary to the existing system on board LUCIE. With the concerns inherent to LUCIE's autonomous execution, the dependable execution of the software on board the excavator can be highlighted as an ideal area for a formal approach to dependable, safe computing to be taken.

For the purposes of formally modelling and ultimately executing code, Petri nets [16] were chosen, not only for their modelling capabilities and the mathematical methods available for testing, but also for the ability to model concurrent programs and allow for a switching of context, as the digger and dumper truck discussed in this paper synchronise and interact with one another.

With the use of Petri nets established as the best method of creating reliable software, it was also necessary to find an equivalent hardware implementation which would not become too expensive, in terms of both time and resources. The Lego Mindstorms kit, in conjunction with the legOS [14] operating system, proved both simple to put into use, and capable of providing the necessary functionality. The Lego Mindstorms RCX boasts not only an excellent interface to its three sensors and three

actuators, but also the capacity to communicate with other RCX robots through an infra-red port. It also features a fully multi-threaded operating system in the form of legOS, a version of C written for the RCX.

3. Design

With the basis of the model's implementation determined, the Petri nets for the digger and dumper truck must then be built up. These were built up initially through the use of the simpler data flow diagram (Figure 3), which allowed for the visualisation of the required functions for the models, prior to establishing the more dynamic Petri net model. The Petri net models were then built up from the functions of the data flow diagram, allowing the states represented by the places to be added once the transitions were already in position, to limit any confusion over the initial modelling process.

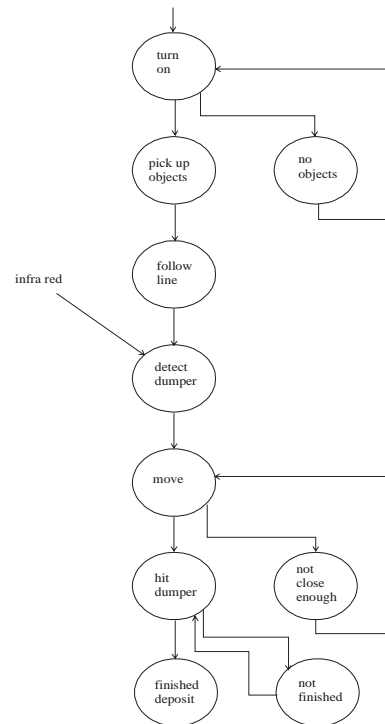


Figure 3 Digger Data Flow Diagram

The modelling process included two separate threads of execution. The first of these was the main Petri net which provided the digger and dumper truck with the means to follow their basic tasks, such as digging, dumping, and following the tracks laid out to get them to their required destinations. This also included a second Petri net component, which was almost completely separate and only activated once the digger detected the

dumper truck (and vice versa) and they had synchronised with one another. The second thread represented the infra-red receive function, a single transition which was required to check for any incoming messages in order to

detect each model's counterpart, and looped continuously until a signal was detected. Once a signal was detected, the colouring of the token could be changed, and the secondary thread of the main Petri net could be activated.

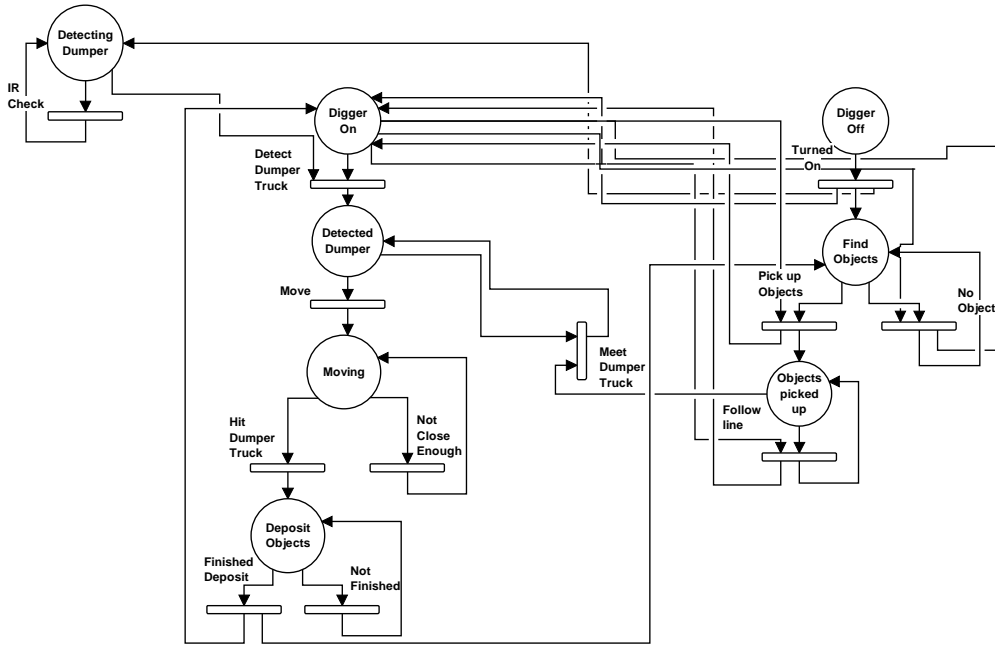


Figure 4 Digger Petri Net

Before any implementation of the models could be carried out, however, they first had to be thoroughly checked. This was done using the TRAMP (Toolkit for Rapid Autonomous Mobile robot Prototyping) [7] toolkit, which would later be responsible for the automatic generation of the software to be executed on board the two robots. The Petri nets were primarily checked for simple reachability requirements, including any colouring restraints, ensuring that the eventual code would reliably reach its intended destination. It was also established that both Petri nets were reversible, ensuring that program could be exited satisfactorily, and it would not be necessary to essentially crash the program in the middle of Petri net execution. This may cause the software to develop memory leaks or become volatile. In fact, in this case it was unnecessary, as the Lego hardware provided adequate back up against these concerns, but reversibility was determined to be a desirable state in principle in the event of execution on board an alternative platform.

4. Implementation

Following the complete testing of the Petri net model, ensuring that the software on board each of the robots would reach all the functions it was expected to, the Petri net was then ready to be converted into legOS C code through TRAMP. This method of converting the model directly into code, prevented any of the model's aspects from getting lost in translation, preserving the structure of the Petri net completely in its new form and maintaining a fault-tolerant software system. However, with the limitations imposed on the digger and dumper truck due to their real-time constraints, it was not possible to model every fine detail of the digger and dumper truck programs within the Petri net. Instead, the code was only partly automatically generated, with the low level detail of each transition implemented by hand.

This hand-coded element may appear initially to re-introduce all the faults that the Petri net model was originally implemented to avoid, including any memory leaks and fatal errors which could occur. However, in fact, the remaining code is entirely segregated into code to be used within individual transitions, which are by definition atomic. It has no external references to any variables, and is restricted to locally defined variables and those passed into the transition as tokens. As such, this code is simple to thoroughly check; for most experienced

programmers a simple glance would probably be enough to verify the code.

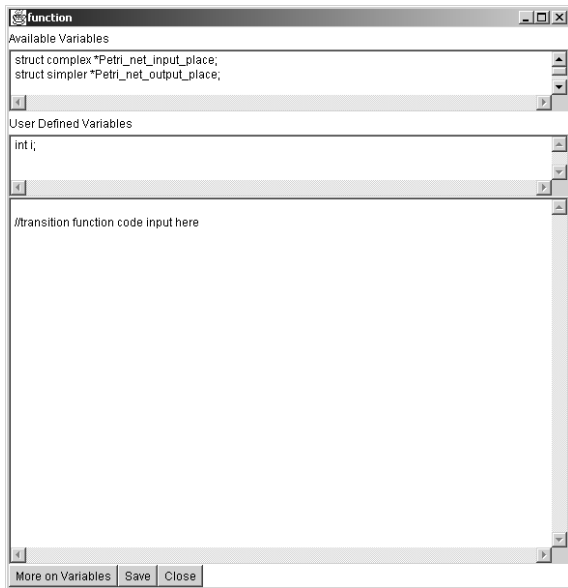


Figure 5 Handcoding a transition

As can be seen in Figure 5, all details relevant to the transition are easily available through TRAMP, and selection of the button “More on Variables” will provide any details on the contents of incoming tokens instantly.

Although the implementation of hardware is not the subject of this paper, it is important to note at this stage a few key elements of the Lego hardware (Figure 6), used to execute the Petri net code produced by TRAMP.

Despite the adaptability of the Lego hardware it was important to make note of its inherent limitations, particularly the sensors available to it – in the form of touch, rotation and light sensors (a camera has since become available). With these limitations in mind, a simple track was devised to allow each robot to follow a line to a particular destination. Detection of the other robot through the infra-red was made as reliable as possible, by positioning the robots close together and facing one another. It was also important that each of the robots maintained certain elements of the behaviours of their real-life counterparts, without becoming too intent on producing precise replicas. For example, the digger performs its digging action with the use of a grabbing

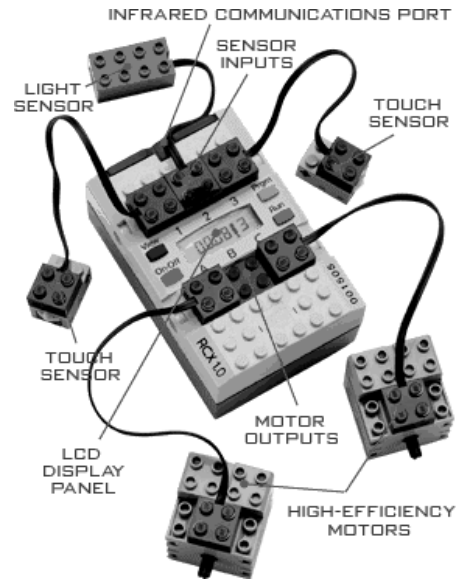


Figure 6 The RCX

motion, producing the same effect for the purposes of this experiment, but completely unrealistic.

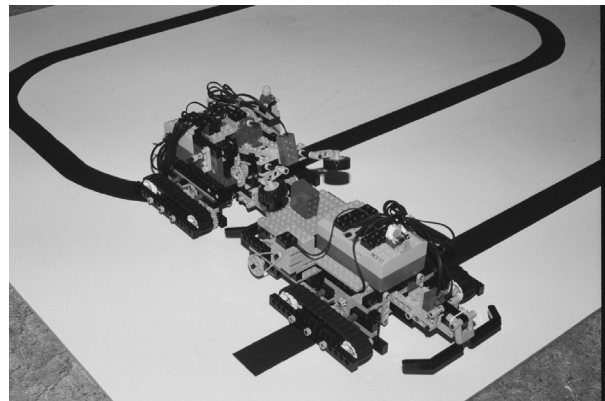


Figure 7 The Digger and Dumper Truck

For the purposes of this experiment, the digger must travel around the circular line (Figure 7), picking up the object en route, then stopping when it detects a wall. Meanwhile, the dumper truck must travel in a straight line along the other line until it detects a wall and turns to face the digger. Once they detect one another through the infra-red detection/transmission, the digger will then turn and dump the object on the back of the dumper truck before restarting its Petri net. The dumper truck must then turn to continue back along the straight line, dumping the object to one side later.

5. Evaluation

Despite the efforts made to limit the impact of the hardware limitations on the overall experiment described above, it was difficult to ignore the main failings of the hardware implementation, by comparison to the software execution. Initial results highlighted the difficulties inherent in any physical simulation, in this case the unreliability of the light sensor on board the Lego RCX, which was required to trace the line on the base, and the infra-red sensor. The infra-red sensor appeared to vary dramatically in its effectiveness, ranging from reflecting off every object in the room, to being unrecognisable from 10cms away. However, these problems were resolved, and the more critical issues of Petri net execution, and reliability of software were then considered.

In light of the numerous hardware problems faced, the software was an unqualified success. There were none of the unexpected and inconvenient problems produced even by this most simple hardware implementation. Instead the Petri net worked perfectly, the only problems within the programming being some early mistakes when setting the colours of tokens, causing transitions never to be enabled despite the availability of incoming tokens. This was in fact possible to test for within the TRAMP toolkit, but has yet to be automatically generated into the Petri net, perhaps an example of the full implications of the ability to automatically generate code from a Petri net without human interference.

It was only in the later stages of implementation that there were any serious software execution difficulties. These turned out to be due to the large increase in memory requirements placed on the RCX by storage of a complete Petri net structure, and gave rise to several instances where the RCX attempted to execute a program without the complete binary available. This was rectified through careful culling of the transition's handwritten code segments, but did successfully highlight one of the main problems with implementation of an automatically generated formal model, and the possibility of a completely dependable program failing despite all efforts.

To give an idea of the scale of the problem, table 1 shows the size of the binaries produced by the digger Petri net and a directly equivalent hand-coded version. It should be noted that the maximum program size permitted on the RCX is 32k, including an allocation for stacks for each thread. Clearly, the saving in removing the Petri net structure is highly significant.

Table 1 Binary Size

Type of Binary	Size of Binary (k)
Automatic digger.srec	31.4
Manual digger.srec	13.0

Traditionally, the other drawback to automatically generated code is the execution time required for all the necessary program management. Although there had been no signs of either the digger or dumper truck performing too slowly, it would be remiss to suggest that there were no overheads produced from the Petri net structure, particularly as the full-sized excavator may come to rely on the speed of the program implementation at some later stage [17]. To establish precisely the nature of the Petri net's overheads, an experiment was performed using a simple one place, one transition Petri net in a loop, against a hand-coded program, each simply adding one to a counter each time the function was entered, until the target was reached. The results in table 2 show the difference in response time between the two to be negligible over 1000 loops.

Table 2 – Execution Times

Automatic	Manual
24.44	24.44
24.45	24.38
24.32	24.46
24.34	24.42

6. Extended Design

6.1 Introduction

In order to expand the concept of concurrency, fault tolerance and communication, an extended design is proposed to identify the area of authentication, to enhance the structure and actions performed within transitions of the Petri net as well as increase the tasks implemented by the digger and dumper truck themselves.

This concept involves the dumper truck authenticating the identity of the digger before allowing the digger to deposit its load. The authentication is based on the Kerberos protocol version 5, as will be seen in section 6.2. Although many different authentication or cryptography techniques could be used within this extended design, Kerberos is used in order to enable authentication of both the client and the server.

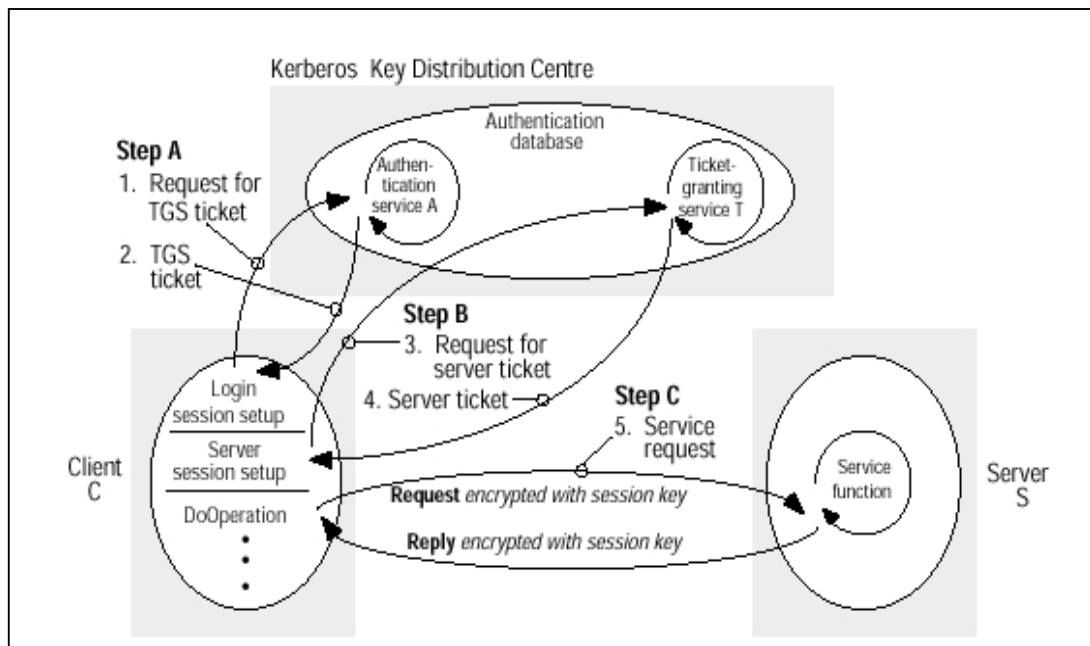


Figure 8 System Architecture of Kerberos [8]

6.2 Background

Security and authentication play a very important role in society today. With the explosion of the Internet and other networked environments, information sent across networks must be protected from the attack of unauthorised personnel. Attacks could come in the form of, for example, eavesdropping, masquerading or tampering; therefore various forms of cryptography must be employed to act as a security mechanism to control the privacy and integrity of the data.

Kerberos is based on the key distribution model [13] and allows a client and server to authenticate themselves with each other, to verify their own individual identities.

Although authentication protocols were developed for use with both public and private (secret) keys, the Kerberos protocol, as discussed within this section uses only secret keys. The key is used to encrypt a *plaintext* message into *ciphertext*, which can then only be decrypted by the decryption key.

Secret key cryptography is symmetric as the keys are only known to the authorised sender and receiver, who must share the knowledge of the key in order for the messages sent between them to be encrypted and decrypted securely, without the risk of attack.

In order to implement this concept the Kerberos protocol version 5 is used to act as an authentication service between the client and the server, which can be seen in Figure 8.

Further information on the Kerberos Protocol can be seen in [8].

Secret keys are used within the Kerberos protocol as the client and server, namely the digger and the dumper truck would only run on a local network as both the robots are contained within a specified area. This would therefore allow a secure key service to be used for key distribution to permit only an authorised digger to deposit its objects into the dumper truck. Only the digger and dumper truck will know the key, therefore the probability of attack is minimal.

Public keys, however do not require a secure key service, using two separate keys to encrypt and decrypt, and therefore are used in large-scale networks such as the Internet. Public keys also only allow one-way communication and are therefore unsuitable for this situation as two way communication must exist between the digger and the dumper truck in order for authentication to occur and the tasks to be completed by each robot to continue.

If multiple diggers were present in the area, each digger must be authenticated individually by the dumper truck using the Kerberos protocol, before the depositing of objects could be initiated.

6.3 Applying Kerberos to the digger and dumper truck

Initially encrypted messages are sent between the authentication server (AS) and the client (the digger) to allow a secret key to be sent to the client in order to begin to prepare for communication with the server (the dumper truck). The authentication server provides a secure means

of obtaining shared keys to allow the communication of separate processes to begin.

The example shown in table 3 identifies the stages involved in authenticating the relationship between the digger and the dumper truck.

Initially the digger receives a ticket and session key from AS to access the TGS, therefore a ticket for client

Table 3 Definitions of abbreviations used within Kerberos authentication

Identity	Key
Client	digger (di)
Server	dumper truck (du)
Key	K
Digger's secret key	K_{di}
Dumper truck's secret key	K_{du}
Secret key shared between digger and dumper truck	$K_{(di, du)}$
Authentication service	AS
Ticket Granting Service	TGS
Nonce	N
Timestamp	T
Start time of valid ticket	t1
Finish time of valid ticket	t2

Any message encrypted in $K_{(di, du)}$ is said to be secure and trustworthy as only the specified digger and dumper truck are able to decrypt the message, having used either the diggers or dumper trucks secret key during encryption. Each message has an expiration time and timestamp encrypted within it to ensure that client-server interaction is only available for a specified period. Each ticket has a lifetime of, in this case, one hour so can be used with other digger-dumper truck communication sessions, however authenticators must be regenerated with each new connection. This ensures that attackers can not impersonate the digger as the ticket becomes worthless after the specified time has expired.

6.4 Construction of authenticating Petri nets

This process is then integrated within the Petri nets for both the digger and the dumper truck. Coloured, timed and hierarchical Petri nets are all used to ensure the main structure of the Petri net is not affected by this authentication and the actual process of authentication and communication does not exceed beyond a specified time period.

Coloured Petri nets are used to allow choices to be made about the authentication during the execution of the Petri net itself. An example is identified in Figure 9, where the TGS server must decide whether the TGS ticket is valid in order for the authentication process to continue. If not, the digger must begin the authentication process

digger (di) to access a server dumper truck (du) is defined as:

$$\{di, du, t1, t2, K_{(di, du)}\}K_{du} \Rightarrow \{ticket_{(di, du)}\}K_{du}$$

Table 4 identifies the various stages involved in forming communication between the digger and dumper truck using the Kerberos protocol.

again, by requesting a TGS ticket from the Authentication Server.

Kerberos involves the use of timestamps to ensure the client-server session only lasts for a specified time period, after which the ticket request process must begin again. This is where t-timed Petri nets are used, not only to ensure the timestamp is not exceeded but also to ensure that the actual stages involved in requesting and receiving the tickets and authenticating the digger with the dumper truck are monitored.

The timings on the transitions as seen in Figures 9 and 10 respectively are set to (0.5, 2) seconds to represent the minimum and maximum length of time required before a transition can be fired. This ensures both that a transition does not take an excessive amount of time to fire and that enough time is allocated for tasks to be completed, therefore allowing tickets and messages to be sent and received between the digger, AS and TGS dumper truck. The timestamp is set to 60 minutes to allow the digger to deposit more than one load into the dumper truck, in addition to avoiding threats from attackers impersonating the digger.

Table 4 – Stages involved in authentication of the digger by the dumper truck using the Kerberos Protocol

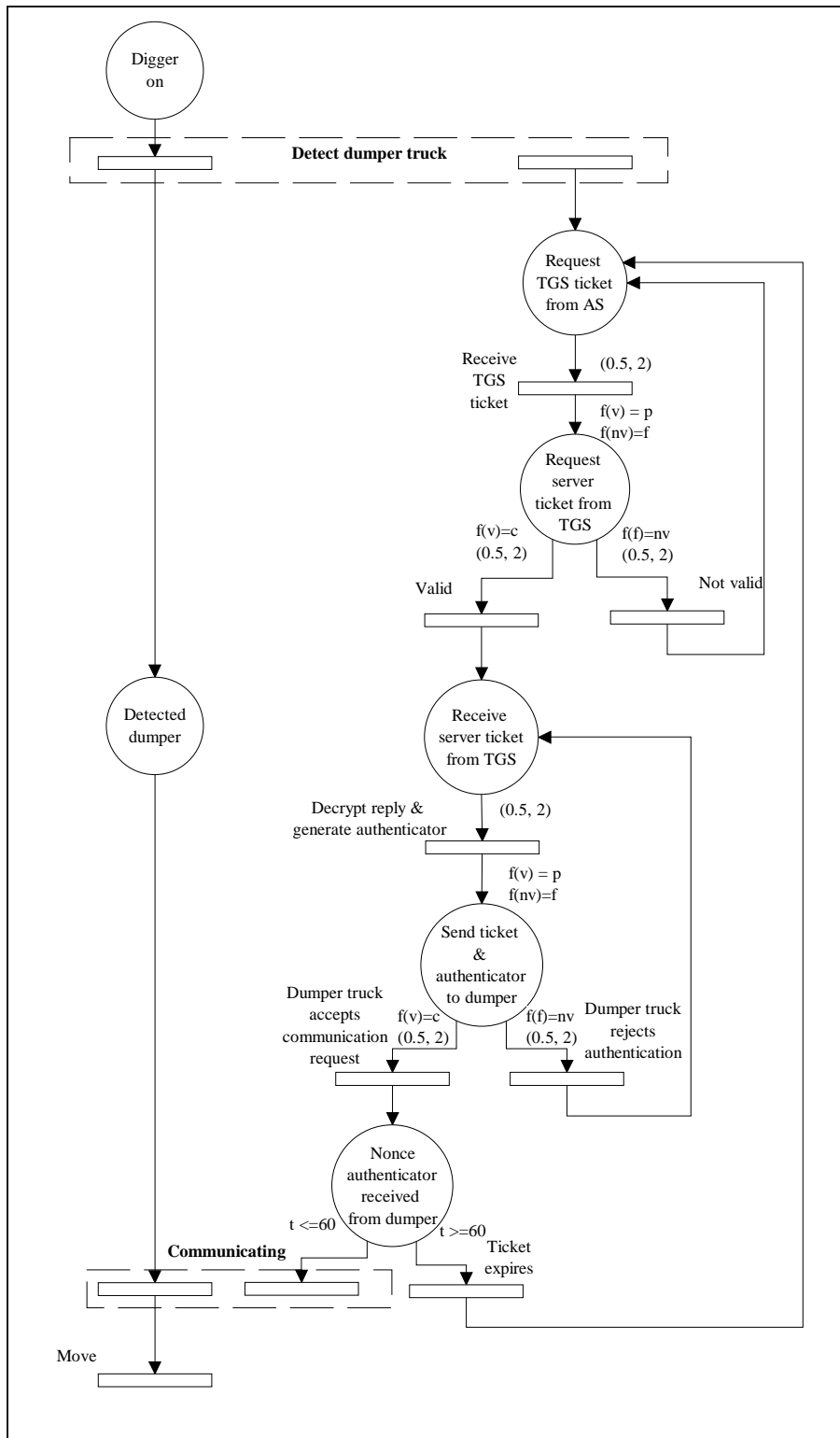
Header	Message	Notes
1) $di \rightarrow AS$	di, TGS, n	Client, di request for the authentication server, AS to generate ticket to communicate with ticket granting service TGS
2) $AS \rightarrow di$	$\{K_{diTGS}, n\}_{K_{di}}, \{ticket(di, TGS)\}_{KTGS}$	AS replies with a message containing a ticket encrypted in its secret key along with a nonce in order to communicate with the TGS .
3) $di \rightarrow TGS$	$\{auth(du)\}_{K_{duTGS}}, \{ticket(du, TGS)\}_{KTGS, du, n}$	di requests that the TGS replies by supplying a ticket for communication with a particular server, du .
4) $TGS \rightarrow di$	$\{K_{di, du}, n\}_{K_{duTGS}}, \{ticket(di, du)\}_{K_{du}}$	TGS verifies that the ticket initially generated by the AS is correct. If valid the TGS generates a new session key with a ticket encrypted in the server's secret key, K_{du} , in order for communication with between client and server to begin.
5) $di \rightarrow du$	$\{auth(di)\}_{K_{di, du}}, \{ticket(di, du)\}_{K_{du}}, request\ n$	Client di sends the ticket to server du , along with an authenticator, used to confirm the identity of the client and a request encrypted in $K_{di, du}$. Authentication of the server is required by the client therefore a request for a nonce to be returned is made, to identify the freshness of the message.
6) $du \rightarrow di$ (optional)	$\{n\}_{K_{di, du}}$	As a request for authenticity is made by the client di , du sends the nonce to di encrypted in $K_{di, du}$.

Hierarchical Petri nets, as described in [9] are not directly implemented within the respective Petri nets. Instead a *plug in module* is inserted into the transitions of the Petri net, requiring a separate thread of execution to be implemented. This allows the actual stages involved during authentication to be identified, without increasing the complexity of the original Petri nets. The contents of the plug in module can be inserted or removed without affecting the structure of the Petri net itself.

The plug in module would be inserted between the transitions “detect_dumper” and “move” in the digger Petri net and between “detect_digger” and “wait” in the dumper truck Petri net, as seen in Figures 9 and 10 respectively, to allow authentication between the robots to be implemented. These threads must begin and end with transitions. Separate tokens could be passed through

this module so as not to interfere with the progression through the structure of the main Petri net and enable complete concurrency in the Petri nets.

After authentication has occurred the digger is able to deposit its objects into the dumper truck, therefore allowing both robots to continue to completion of their own individual tasks.



Key

AS = Authentication Server

TGS = Ticket granting Service

Token Colour Key

v = valid

nv = not valid

p = pass

f = fail

c = continue

Timing Key

t = time

(0.5,2) = min and max time

60 mins = length of timestamp

Figure 9 – Inserting a dead reckoning module within the digger Petri net

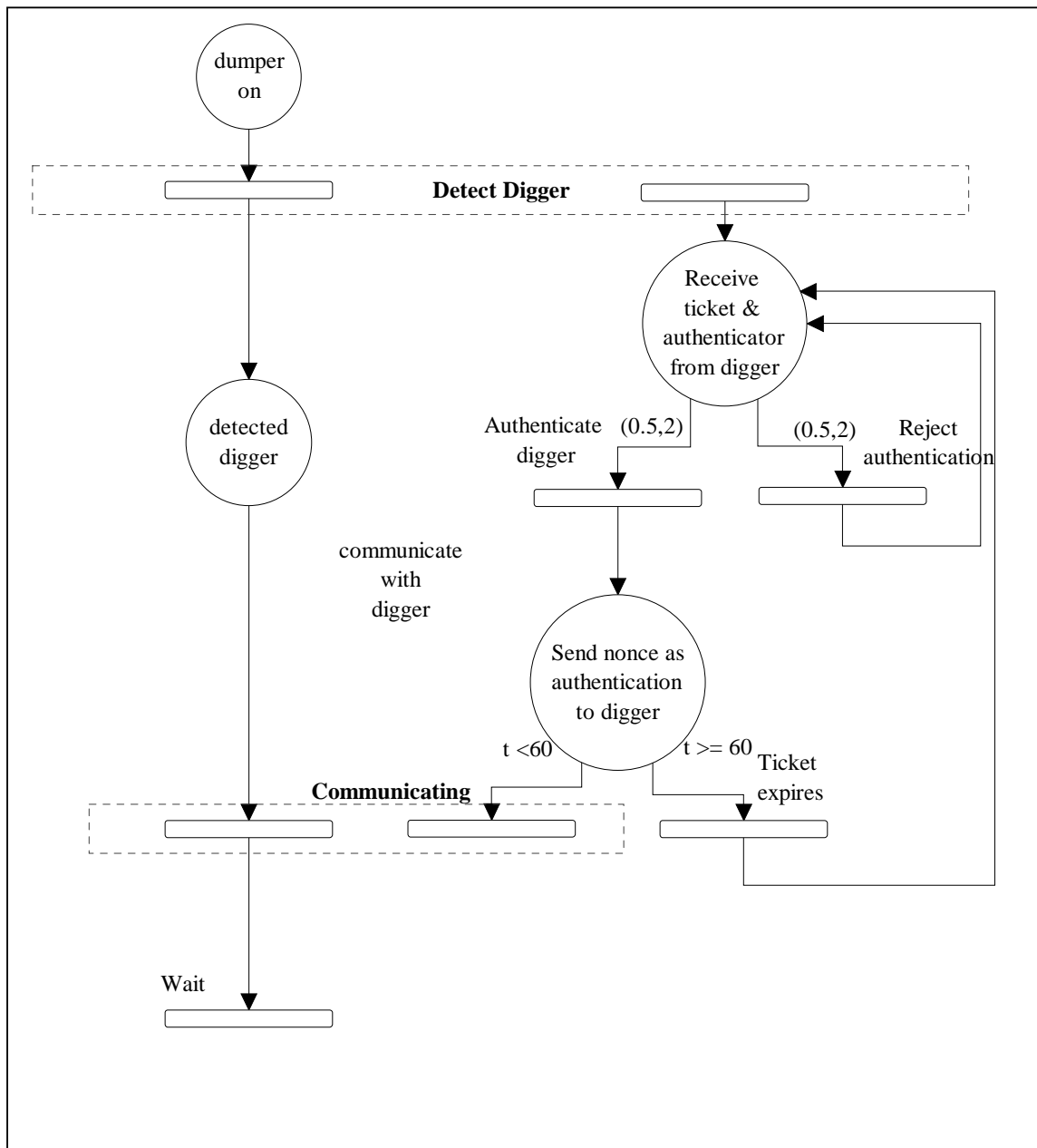


Figure 10 – Inserting a dead reckoning module within the dumper Truck Petri net

To summarise, the Kerberos protocol is intended to allow the dumper truck to authenticate the digger, before the digger is able to deposit its load. Encrypted messages are initially sent between the *authentication server (AS)*, situated within the *key distribution centre*, and the *client* (the digger) in order for tickets encrypted in the secret key to be sent to the various services within Kerberos. This enables communication between the digger and dumper truck to begin. After a new session key and ticket, encrypted in the *server's* (the dumper truck) secret key, has been generated by the *ticket granting service*

(*TGS*), the digger may begin its communication with the dumper truck, provided authentication has been granted. This will be achieved by inserting a plug in module between the “detect_dumper” and “move”, and “detect_digger” and “wait” transitions in the digger and dumper truck Petri nets, respectively.

After authentication and communication between the digger and dumper truck has occurred, each robot is able to continue with their specified tasks and complete the atomic actions described within the transitions of the Petri net.

7. Conclusions and Future Work

In terms of overall dependability, it can be concluded that the concurrent program, generated through the Petri net for both the digger and the dumper truck was for the most part successful. It completed the tasks set before it with only minor difficulties, largely caused through mechanical failure, and demonstrated the use of the software adequately. The execution of the Petri net was also efficient enough that the real-time requirements of the system can be considered to have been met, further encouraging the development of the methods involved. Despite this, there is still a way to go before the system could be considered completely dependable. The few minor problems, such as the failed colouring of the tokens could be catastrophic should they occur on board a full-size excavator, or even a larger model, and it is by these terms that any ultimate success or failure must be measured.

It should also be noted, that the legOS C language was chosen over any versions of java [11] in part because of the additional memory that running a Java virtual machine and writing object-oriented code requires. This system was intended to be useable on board systems with limited memory requirements and as such a degree of failure must be accepted due to the huge excess of memory taken up by the Petri net structure and management.

It is in these two areas, of token colouring and memory usage, that any future work should be targeted. For the purposes of colouring the tokens, the TRAMP toolkit is already in a position to simulate this colouring, and so the option of partially automating the final executable code is already a possibility. However, the difficulties faced with memory usage remain, and without further work in this area, it will soon be impossible to run code of any complexity on board an RCX, or equivalent system. This is a problem which must be faced in the immediate future, and, if necessary, a decision taken on whether to maintain complete Petri net structure, or to minimise the cost to memory.

Finally, it would be of interest to fully implement the Kerberos protocol within the Petri net plug-in. This extension is as yet some time away, but the mechanisms are in place to provide for this addition to the Petri net, so in the future this is likely to emerge.

8. References

- [1] Dave Baum's Definitive Guide to Lego Mindstorms. Baum D. Apress 1999.
- [2] Developing real-time autonomous excavation – the LUCIE story. Bradley DA, Seward DW. Proceedings of the IEEE Conference on Decision and Control, 1995, Vol 3, p 3028-3033
- [3] A hierarchical View on GCSPNs and its Impact on Qualitative and Quantitative Analysis. Buchholz P. Journal of Distributed Computing, 1992, Vol 15, pp 207 – 224
- [4] A Technique for Designing Robotic Control Systems Based on Petri Nets. Caloini A, Magnani G, Pezze M. IEEE Transactions on Control Systems Technology, Vol 6, No 1, pp 72-87. 1998.
- [5] Impact of Fieldbus on Communication in Robotic Systems. Cavalieri S, DiStefano A, Mirabella O. IEEE Transactions on Robotics and Automation, 1997, Vol 13, No. 1, pp 30-48
- [6] An Object-Oriented Petri Net Toolkit for Mechatronic System Design. Chandler A. PhD Thesis, Lancaster University, UK 1999.
- [7] Testing Petri Nets for Mobile Robots Using Gröbner Bases. Chandler A, Heyworth A, Blair L, Seward D. 21st International Conference on Application and Theory of Petri Nets: Software Engineering and Petri Nets Workshop Proceedings (2000), p21-34.
- [8] Coulouris, Dollimore, Kindberg (2000) Distributed Systems Concepts and Design (3rd Edition) (Addison-Wesley Publishing Company)
- [9] Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use Volume 1. Jensen K. Springer-Verlag. 1997.
- [10] Lego Mindstorms. <http://www.legomindstorms.com/>
- [11] <http://lejos.sourceforge.net/>
- [12] Hand-in-Glove Human-Machine Interface and Interactive Control: Task Process Modelling Using Dual Petri Nets. Mascaro S, Asada HH. Proceedings - IEEE International Conference on Robotics and Automation, 1998, Vol 2, pp 1289-1295
- [13] Using Encryption for Authentication in Large Networks of Computers. Needham RM, Schroeder MD, 1978. Comms, ACM, vol 21, p993-9.
- [14] LegOS. Noga M. 1999. <http://www.noga.de/legOS/>
- [15] Development of a Safety Manager for an Autonomous Mobile Robot', C. Pace, D.W. Seward, Proceedings of the 29th International Symposium on Robotics 1998 (ISR98), p 277-282
- [16] Petri Net Theory and the Modelling of Systems. Petersen JL. Prentice-Hall 1981.

- [17] Efficient Automatic Code Generation for Embedded Systems. Pilaud D. Microprocessors and Microsystems (1997), v20, n8, p501-504

- [18] LUCIE the robot excavator – design for system safety. Seward D, Margrave F. Proceedings – IEEE International Conference on Robotics and Autonomy, 1996, p 963-968

- [19] Design and Analysis of Synchronisation for Real-Time Closed-Loops Control in Robotics. Simon D, Castaneda EC, Freedman P. IEEE Transactions on Control Systems Technology, 1998, Vol 6, No. 4, pp 445-461

- [20] Design of a Supervisory Control System for Multiple Robotic Systems. Suh IH, Yeo HJ, Kim JH, Ryoo JS, Oh SR, Lee CW, Lee BH. IEEE International Conference on Intelligent Robots and Systems, 1996, Vol 1, pp 332-339