A pervasive design strategy for distributed health care systems

Oliver Faust[*1], Bernhard H.C. Sputh[1], Rajendra Acharya U[2], Alastair R. Allen[1]

[1] *School of Engineering, University of Aberdeen, Aberdeen AB24 3UE, UK*
[2] *Ngee Ann Polytechnic, Clementi Road 535, Clementi, Singapore*

Abstract: Distributed health care systems require a pervasive design strategy to ensure security, stability and functionality of the system. Furthermore, distributed health care systems are potentially very complex, which makes it necessary to have appropriate abstraction strategies. In this paper we introduce the theory of communicating sequential processes to distributed health care system design. We use this theory to gain further insight into the communication between different system components. In particular this reasoning is applied to the problem of internal and external integration of distributed health care systems. We claim that a high level of internal and external integration can be achieved with the pervasive design strategy. With the pervasive concept of communicating sequential processes we have a formal method to create a model of the complete system. In the practical part of the paper we apply the formal system model to prove security, stability and functionality of a particular system.

Keywords: Formal system design, Distributed health care, Communicating sequential processes

1. INTRODUCTION

The field of health informatics (eHealth) has mushroomed since its inception in the early 1990s. It is concerned with the cognitive, information processing and communication tasks of medical practice, education and research [1]. One of the reasons for this astonishing growth is that eHealth has the potential to improve the quality of care, whilst simultaneously lowering costs for complex cases [2].

There are numerous projects and studies which are exploring this potential. Many of these projects are carried out in developed countries, therefore these projects address current needs of these relatively wealthy nations. A typical example is the epidemic of human overweight and obesity. Sunny Consolvo *et al* address this problem by investigating how technology could help encourage people to sustain an increased level of physical activity [3]. They conclude that mobile phone technology can be used in systems which attempt to solve this problem. They presented Houston, a mobile phone-based fitness journal that encourages physical activity by providing personal awareness of activity level and mediating physical activity-related social interaction among friends. However, Houston is, by far, not the only mobile phone based health care system. Tammy Toscos *et al* developed a preventative health cell phone application that helps to motivate teenage girls to exercise by exploiting their social desire to stay connected with their peers [4]. Their aim was to create a system to reach out to young women who enjoy technology and need motivation to continue being physically active throughout adolescence. In a similar project, Gunny Lee *et al* created PmEB, an application for mobile phones that allows users to monitor their caloric balance as a part of weight management [5]. PmEB addresses the challenge of improving user compliance, data reliability, and self-efficacy of self-monitoring of calorie intake and expenditure. Related to the problem of overweight and obesity is the implementation of PREVENIMSS, a Spanish national program for education and prevention of diseases caused by malnutrition and obesity by Juan M. Silva *et al* [6]. PREVENIMSS works by organizing informational sessions about the risks of keeping sedentary life and bad feeding habits.

Apart from these relatively targeted projects there are developments which aim at general heath care. Misook Sohn and Jeunwoo Lee used Instant Messaging systems to share context information relating to health and covering things such as physical activity and smoking behaviour [7]. Toshiyo Tamura *et al* developed a fully automatic healthcare system for use in a so-called welfare techno house. The system automatically collects physiological data in an effective and controlled way [8]. They claim that their system helps in the understanding of personal health status and daily activity without the use of invasive measurements. Similarly consumer or patient related is the work by Diana Bental and Alison Cawsey. They developed a range of systems that synthesize text to speed up content creation and access. For example, the program 'mail merge' was used by health professionals to produce printed letters and leaflets that incorporate information for specific individuals [9]. Within the group of general heath care systems many are aimed at professionals. K. A. Kuhn and D. A. Giuse claim that hospital information systems are evolving towards health information systems [10]. Terry L. Huston and Janis L. Huston argue that today, health-care decision-makers are struggling with the ramifications of the managed care environment [11]. Concurrently, technological advances take place that create a benevolent convergence with the upheaval generated by the managed-care movement. John Fulcher claims that a major conclusion of his study was that so-called high-tech solutions are not always necessary or effective for health care systems [12]. This claim is based on the fact that if patient privacy, confidentiality and security are not incorporated from the outset, then this can lead to unimagined difficulties at some future time — indeed such systems can be rendered inoperative. These considerations lead to yet another set of problems heath

*Address correspondence to this author at the Electronics Research Group (ERG), Department of Engineering, University of Aberdeen, Fraser Noble Building Aberdeen, Scotland, AB24 3UE; United Kingdom; E-mail: o.faust@abdn.ac.uk

1874-4311/08

care professionals face. From a computer science point of view health information systems are huge data gathering devices. Data gathering implies data storage and data transmission. It is necessary to safeguard against deliberate misuse or loss of data. Boaz Gelbord and Gert Roelofsen examined the use of computers to extract information from large amounts of data with regard to privacy and public safety. They show that due to the use of computers to extract information from data sources, such as video surveillance recordings, all collected data is sensitive [13].

The root of the problems and indeed their solution lies in the fact that telemedicine and eHealth both depend on communication technologies [14]. Wullianallur Raghupathi and Joseph Tan claim that the success of IT based health care systems depends on [1]:

- Internal integration. – The degree to which systems and technologies are integrated with one another within an organisation.
- External integration. – The degree to which systems and technologies interface with outside organisations and agency computer systems.

In this paper we interpret internal and external integration in terms of network terminology and process orientation. We claim that systems which are designed according to process orientation reach a high level of integration. We are even able to extend the requirements of internal and external integration when we say that this is the functionality of the system, and functionality is only one part of system specification. System specification defines the system properties in terms of security, stability, and functionality. These properties are the fundamental requirements for intelligent distributed diagnosis and home healthcare systems, because such systems are designed as networked components which work cooperatively on the decision making task.

1.1. Towards Process Orientation

To achieve the goal of high internal and external integration we adopt the theory of communicating sequential processes (CSP). This theory was first introduced by C. A. R. Hoare [15]. By now, CSP has over 25 years of solid research behind it, see [16]. During this time it has been applied to diverse topics such as hardware synthesis [17], concurrent programming [18], security [19] and systems testing [20]. This flexibility makes CSP ideal for modelling distributed diagnosis and home healthcare systems. CSP enables the formal specification of such systems and their refinement to executable implementation. It allows the description of complex patterns of synchronisation between component processes and provides semantics sufficiently powerful to capture non-determinism, multiway synchronisation, channel communication deadlock and divergence [21]. We link CSP and distributed diagnosis and home healthcare systems by modelling each component as a process and the communications between components (processes) as events. To provide another glimpse of the theory, the following list defines the system properties according to CSP principles and links them with distributed health care:

- Security – A system is secure if and only if it is not able to exhibit a potentially hazardous sequence of events (traces). A distributed health care system is secure if it exhibits only allowed sequences of events. For example, only authorised persons should be abele to access data. Security is bridged if unauthorised people can gain access. In terms of CSP this security bridge can be stated as follows: unauthorised people took some action which generated a sequence of events for the system. The system responded to this sequence of events by releasing the sensitive data. Clearly, the system reacted in the wrong way to this particular sequence of events, therefore it is not secure.
- Stability – A system is stable if it does not have any deadlock or livelock conditions. For distributed health care systems stability is the second most important property after security. This statement is true, because a failed system does not exhibit any event at all, i.e. no sign of functionality at all. This implies that the system is not able to exhibit a potentially dangerous event, therefore it is secure. For example, a distributed health care system which incorporates the *MAILER* system, discussed in Section 3, will show a stable failure. To be specific, the system can be forced into deadlock by a large number of communication requests. This is potentially very dangerous in emergency situations, because such situations usually produce a large number of communication requests and many people depend on the stable function of the system.
- Functionality – A system is functional if it behaves according to the specification. This is measured by comparing the specified sequences of events with the sequences of events a system can exhibit. Functionality is what we want from a distributed health care system. We have to lay down a set of rules (specification) which govern the behaviour of the system. A distributed health care system is functional if and only if it complies with these rules. CSP helps to overcome the practical difficulties in laying down the rules and checking for compliance.

These definitions provide a reference for the analysis of system properties. The act of analysing a system with respect to security, stability, and functionality, is called model checking. If model checking is carried out in a formally correct way, it leads to proven statements about the system properties. For example, to make a statement about the security of system P, we have to identify a set of hazardous traces H. And then, in a second step, the set S with all the possible traces of the system is computed. The system is secure, if and only if there is no trace x which is an element of the hazardous set H and an element of the set of all possible traces S. We can define a function secure(...) which returns true if the *process* is secure. (For a discussion of the mathematical symbols see [22].)

$$\text{secure}(H, S) = \neg \exists x \bullet (x \in H \wedge x \in S) \qquad (1)$$

The big benefit of having such rigid definitions and the associated CSP theory is that model checking can be mechanised. To follow the security example, it is possible to use model

checking tools to compute all possible traces of a process, see [23].

The pervasive design strategy for distributed health care systems is a three step process. The first step is the development of a theoretical CSP model which serves as system specification. The next step is to check the specification for security, stability and functionality. Automated model checkers, such as FDR, can be used for this task. The last step of the pervasive design strategy is the translation of the specification into an implementation. Figure (1) illustrates the proposed design flow.

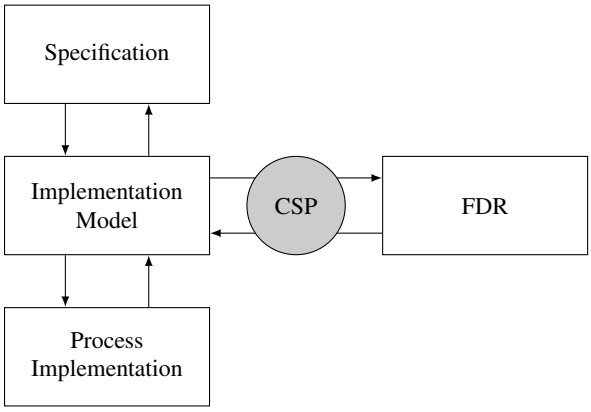

Fig. (1). Overview of the pervasive design strategy

Before we detail the example system, the next section provides some background on the process algebra CSP. The section does not introduce all theorems and lemmas which arise from the axiomatic definition of the process algebra, it just provides some simple examples which explain important concepts. These important concepts are deadlock, divergence (livelock), choice and abstraction. Based on these examples we introduce mechanised model checking. This leads to considerations about failures divergence refinement. Discussing the CSP method, we move on to the main design exercise. Distributed diagnosis and home healthcare systems require a routing network to exchange data between individual entities. This is the reason why the design exercise is a distributed mailer system. The design follows the approach outlined in Figure (1). Section 3 outlines the basic mailer process network. However this process network is not stable, i.e. there is a deadlock condition. With mechanised model checking we show that when all connected nodes want to send a message, the mailer deadlocks! To avoid this deadlock, Section 3.4 introduces an arbiter which ensures that only one message is in the mailer network at the same time. The paper finishes with concluding remarks and further work.

2. THE LINK TO CSP

This section provides a brief introduction to CSP. We present a number of examples which explain the fundamental principles of the CSP process algebra. This theory is necessary to understand the benefits of the pervasive design strategy for distributed health care systems. The examples can not replace an in-depth treatment of the process algebra, Schneider and Roscoe provide such treatments in their books [24], [22].

The examples are abstract rather then practical, because the concepts are best introduced without the burden of practical considerations. The first of these concepts is deadlock. The deadlock example allows us to introduce the concepts of events, processes and synchronisation. Then choice is introduced. We show that in CSP there are two types of choice, namely external and internal choice. The next concept which we introduce in this section is non-determinism. Abstraction and hiding extend the concept of non-determinism. A non-deterministic system might livelock.

Livelock and deadlock constitute the two stable failures which are relevant for distributed health care systems. Mechanised model checking helps to detect stable failures and establishes system security and functionality. Such mechanised model checking is introduced based on FDR (Failures Divergence Refinement). Model checking does not guarantee that the implementation is correct, however it guarantees that it is possible to build or implement a system which is secure, stable and functional.

2.1. Deadlock

In the case of distributed health care systems, the term deadlock describes the situation when two or more independent processing entities prevent each other from making progress. The simplest example involves two processing entities ($P1$ and $P2$) and a duplex message transfer channel between them. At one stage of the execution $P1$ is determined to send a message to $P2$ before it can make further progress. A deadlock occurs if $P2$, in the same state of execution, is determined to send a message to $P1$ before it can make progress. In this case, deadlock means $P1$ does not progress because it waits for a message from $P2$ and at the same time $P2$ does not progress because it waits for a message from $P1$.

To model this deadlock scenario with CSP, we need three processes, two of which represent the processing entities. The third process represents the environment in which the messages are exchanged. The communication between the processing entities is modelled as the exchange of events between processes. In this case the message from process $P1$ to process $P2$ is represented by a and the message from $P2$ to $P1$ is represented by b.

We model the functionality of the deadlocking system by describing what each involved process does. Initially, $P1$ is able to perform event a then a again and then b before it terminates. The previous sentence communicates the underlying idea of the model. However, text is not formal enough to establish provable statements about security, stability and functionality. The CSP process algebra and the associated operational semantics provide the required level of formality. The following equation states the operational semantics for the $P1$ process.

$$P1 = a \rightarrow a \rightarrow b \rightarrow SKIP \tag{2}$$

where $\rightarrow$ indicates a transition or change of state and *SKIP* represents successful termination. $P2$ is initially able to perform a then b then a again and so on. The following recursive

expression describes this functionality.

$$P2 = a \rightarrow b \rightarrow P2 \tag{3}$$

The $P3$ process describes the environment in which $P1$ and $P2$ communicate. We say that $P1$ and $P2$ must synchronise on the events a and b. The following equation models this behaviour.

$$P3 = P1 \underset{\{a,b\}}{\parallel} P2 \tag{4}$$

where $\parallel_{\{a,b\}}$ is the indexed parallel operator. The set $\{a, b\}$ below the parallel operator forces processes $P1$ and $P2$ to synchronise on the events a and b. Synchronisation means that these events can happen if and only if both processes $P1$ and $P2$ are able to perform them. Initially, $P1$ is able to perform a and $P2$ is able to perform a. That means this event can and will happen. Next, we consider the situation or system state when the event 'a' has happened. Now, $P1$ wants to perform a again, but $P2$ is only able to perform b. In other words, the parallel statement forces $P1$ to wait until the communication partner $P2$ is able to perform a. Similarly, the same synchronisation statement forces $P2$ to wait until b can be performed. This is a classical deadlock situation, the two communication partners prevent each other from making progress.

2.2. External choice

In many natural situations there is a choice between two or more possible actions. In CSP external choice describes the situation when a system accepts any one of a number of offered actions. The environment is in control, because it decides which action to take. Therefore, for the environment such a choice is deterministic: the system accepts any of the offered actions. The process $P4$, defined in Equation 5, describes such a situation. This process is able to perform either a or b before the process recurses.

$$\begin{aligned} P4 \quad = \quad & a \rightarrow P4 \\ & \Box \ b \rightarrow P4 \end{aligned} \tag{5}$$

where $\Box$ is the external choice operator, with which the environment can choose whether a or b occurs.

The process $P5$ is the environment in which $P4$ and $P2$ communicate. Equation 6 defines $P5$ as the parallel combination of $P4$ and $P2$. The parallel processes must synchronise on the events $\{a, b\}$.

$$P5 = P4 \underset{\{a,b\}}{\parallel} P2 \tag{6}$$

Within $P5$ the process $P2$ constitutes the environment which chooses whether a or b occurs. From the definition of $P2$ in Equation 3, it follows that $P5$ exhibits events a then b then a again and so on. That means, $P5$ behaves similarly to $P2$, or in other words the parallel combination of $P2$ and $P4$ does not restrict the functionality of $P2$.

2.3. Internal choice

Internal choice models the situation when the process is in control. That means, the system selects the action to take. This makes a system with internal choice non-deterministic for the environment. The operational semantics in Equation 7 describe such a situation. Process $P6$ decides whether it engages in event a or b.

$$\begin{aligned} P6 \quad = \quad & a \rightarrow P6 \\ & \sqcap \ b \rightarrow P6 \end{aligned} \tag{7}$$

where $\sqcap$ is the internal choice operator.

The process $P7$ defines the parallel combination of $P6$ and $P2$. The combined processes must synchronise on the events $\{a, b\}$.

$$P7 = P6 \underset{\{a,b\}}{\parallel} P2 \tag{8}$$

Within $P7$ the process $P2$ constitutes the environment for $P6$, the process with the internal choice. Equation 3 defines that $P2$ exhibits a then b then a again and so on. The process $P6$ chooses internally in which event, either a or b, it is able to engage. In the initial state of execution $P2$ is able to exhibit a and $P6$ chooses internally whether or not a can happen. If $P6$ chooses b in the initial state then the system deadlocks.

2.4. Abstraction and hiding

Abstraction reduces the model complexity. This leads to simpler models which are easier to understand and to predict. The idea is to represent complex systems with somewhat simpler models that capture the essence or main functionality of the complex system. In CSP abstraction is achieved with the hiding operation. The hiding operation makes events internal to processes, such that an external observer is unable to observe the hidden events.

For example, the process $P8$, shown in Equation 9, is constructed such that it initially offers the environment the choice between event c and d. However, c is hidden from the environment with the hiding operator: '$\setminus$'. Therefore, the environment is not able to exercise this choice. The occurrence of c is non-deterministic, because it is internal to $P8$ i.e. it does not depend on external influences. Therefore, for the environment it appears that $P8$ chooses internally between a and b.

$$P8 = \left(\begin{array}{l} c \rightarrow a \rightarrow P8 \\ \Box \ d \rightarrow b \rightarrow P8 \end{array} \right) \setminus \{c, d\} \tag{9}$$

In Equation 9 the hiding operation converts an external choice into an internal choice. This observation leads us to two conclusions. First, the hiding operation reduces the information content of a system, it becomes less deterministic. Second, non-deterministic operations model abstracted behaviour.

2.5. Livelock

Livelock describes the situation when a system makes progress without engaging in external events. That means, an external observer does not observe any events, but the system consumes energy. The process $P9$, defined in Equation 10, describes such a situation. This process might decide to engage only in event a. If this is the case an external observer is unable to observe any event, because a is hidden with the hiding operator.

$$P9 = \left(\begin{array}{l} a \rightarrow P9 \\ \sqcap \ b \rightarrow P9 \end{array} \right) \setminus \{a\} \tag{10}$$

2.6. Model checking

The previous sub-sections introduce CSP operational semantics. These semantics provide a formal way to model the functionality of distributed health care systems. The formal specification allows us to prove certain properties of the model. Furthermore, this reasoning can be mechanised, this leads to automated model checking. FDR* is a tool which provides such mechanised model checking. In this section we use this tool to check the examples for deadlock and livelock conditions. Furthermore, we use the tool to introduce trace and stable failure refinement.

The first step towards automated design checking is to transfer the operational semantics into a machine readable format. The result of such a translation is a CSP_M script. Listing 1 constitutes the CSP_M version of the examples defied in the previous section. The code in Line 1 defines all possible events. The code block from Line 3 to 10 defines the example processes. The last part of the script, Lines 12 to 23, instructs the model checker to carry out various tests.

```
1  channel a,b,c,d

3  P1 = a -> a -> b -> SKIP
   P2 = a -> b -> P2
5  P3 = P1 |[{a,b}]| P2
   P4 = a -> P4 [] b -> P4
7  P5 = P4 |[{a,b}]| P2
   P6 = a -> P6 |~| b -> P6
9  P7 = P6  |[{a,b}]| P2
   P8 = (c -> a -> P8) [] (d->b -> P8)
11

   assert P3 :[ deadlock free [F] ]
13 assert P5 :[ deadlock free [F] ]
   assert P7 :[ deadlock free [F] ]
15 assert P6\{a} :[ deadlock free [F] ]
   assert P6\{a} :[ divergence free ]
17 assert P6 [T= P4
   assert P4 [T= P6
19 assert P4 [F= P6
   assert P8\{c,d} [T= P6
21 assert P6 [T= P8\{c,d}
   assert P8\{c,d} [F= P6
23 assert P6 [F= P8\{c,d}
```

Listing 1 CSP_M source code for the example processes

Figure (2) shows the results of the automated checks. A red '×' indicates a test was not successful and green '✓' indicates a test was successful. The stability checks, Lines 12 to 15 hold no surprises. The tool confirms all the statements, we made in the previous sections, about the process stability.

In Listing 1 Lines 17 to 23 instruct the FDR tool to perform refinement checks. The first of these refinement checks is trace refinement. A trace is a sequence of events which can be detected by an external observer. The traces of a process is the set of all possible traces this particular process can exhibit. The successful trace refinement check in Line 17 established that the traces of `P4` are a subset of the traces of `P6`. Similarly, the next line of trace refinement establishes that the traces of `P6` are a subset of the traces of `P4`. That means, `P6` and `P4` are trace equivalent. However, the failure refinement check in Line 19 fails. That means, internal and external choice are not the same, they have different stable failures. The remaining four refinement checks establish that `P6` and `P8` have the same traces and the same stable failures. Therefore, there is no difference between `P6` and `P8`. This supports the statement that abstraction can convert external to internal choice, thus making the resulting process less deterministic.

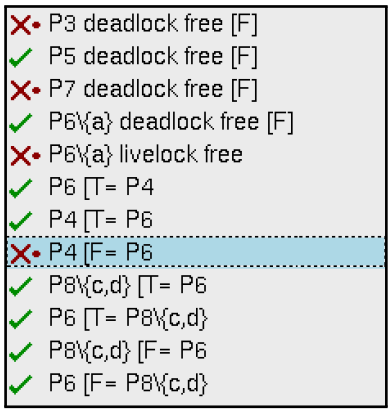

Fig. (2). Output of the model checker tool FDR.

3. MAILER

As outlined in the introduction, distributed health care systems monitor health parameters and interact with patients. To achieve this functionality the individual system components must exchange information via a routing network. The following list outlines three practical distributed health care systems where we stress the central role of the routing network:

1) Blood Pressure – Subjects with high blood pressure need to monitor their blood pressure. The blood pressure measurements are transferred via a routing network to a central processing system. The software which runs on this central processing system will interact with the patient. To be specific, it will prescribe them the diet and encourages them, if there is a decrease in the blood pressure. If there is an increase in the blood pressure it cautions them.
2) Cardiac health – Subjects with cardiac abnormalities can send their ECG through the internet to a server for the physician to inspect. This helps the cardiologist to monitor their patient's cardiac state using his unique ID. In this example the internet represents the routing network.
3) Medication adjustment – Physicians can access his patient's images from a remote place to constantly monitor

*From Formal Systems Europe

the efficacy of the drug and progress of the disease. The images travel over a routing network.

After having stressed the importance of such routing networks, we highlight potential design shortcomings in these structures. A routing network enables independent entities to send messages to one another. Such systems can suffer from security, stability and functionality problems. For example, a stability problem can arise during mass casualty incidents [25]. In such incidents, a potentially enormous amount of data, including blood pressure and cardiac abnormalities, are gathered and communicated in distributed health care systems. It is possible to overwhelm these systems, when they are not designed properly [26].

We argue that a data handling architecture must take into account such extreme situations. To support this statement with a practical example we introduce the mailer system, a data handling architecture. It receives a message from one input and sends it to a distinct output. We achieve this functionality with a routing network where each network node can act as both message source and destination. The network nodes communicate with a packet based protocol. Each packet consists of message and destination, where the destination is the ID of the node which is supposed to output the message.

In the following sub-sections we use our pervasive design strategy for distributed health care systems to plan the mailer system. In the first step we develop the functional model. Then we establish security, stability and functionality properties for this model. The functionality of the theoretical models in the introduction was straightforward. We had a good control and could detect stable failures before hand. This is different for the mailer system, we discover that a straightforward approach to the mailing problem results in a hidden deadlock condition. This is a design error. Past experience shows that in classical approaches design errors are hard to detect. To be specific, it is hard to detect them early within the project. With the mailer example we can show how the CSP approach improves the understanding of the system which leads to improvements in functionality and stability.

3.1. Distributed mailer

Figure (3) on Page 64 shows the process network of a distributed mailer system. It consists of 2^n independent *NODE*(*ID*) processes. The process ID is encoded in an n dimensional binary number. Each *NODE*(*ID*) process is connected to an internal routing network and individual input and output channels. In the process network diagram, shown in Figure (3), a channel which belongs to the internal routing network is depicted as  and labelled as *c.from.to* where *from* indicates the sender *NODE* and *to* indicates the receiver *NODE*. The two arrow heads indicate a bidirectional link, therefore each channel which belongs to the internal routing network has two labels. The individual input and output channels are depicted as  and labelled: *in* and *out*.

Apart from the general network setup, Figure (3) shows also a particular example of a packet being routed through the network. We use this example to explain the mailer algorithm functionality. In the example the environment sends the packet $\langle\langle 1,0,0\rangle, 42\rangle$ over the channel *in* to the *NODE* with the ID $\langle 0,1,1\rangle$. The address of the packet is $\langle 1,0,0\rangle$ and the message is 42. After having received the packet from the environment, *NODE*$(\langle 0,1,1\rangle)$ recognises that the first entry in the address vector is different from the node ID. Therefore, it sends the packet to *NODE*$(\langle \text{not}(0),1,1\rangle)$. In other words the first entry of the *NODE* ID is inverted. Sending the packet results in the event:

$$c.\underbrace{\langle 0,1,1\rangle}_{from}.\underbrace{\langle 1,1,1\rangle}_{to}.\underbrace{\langle\langle 1,0,0\rangle,42\rangle}_{packet} \tag{11}$$

After *NODE*$(\langle 1,1,1\rangle)$ has received the packet, it uses the same method to determine the next node address. To be specific it recognises that the second entry of the address is different from the *NODE* ID. Therefore, the following event is created:

$$c.\langle 1,1,1\rangle.\langle 1,0,1\rangle.\langle\langle 1,0,0\rangle,42\rangle \tag{12}$$

Similarly, after *NODE*$(\langle 1,0,1\rangle)$ has received the packet, it is sent on to *NODE*$(\langle 1,0,0\rangle)$. This is done with the following event:

$$c.\langle 1,0,1\rangle.\langle 1,0,0\rangle.\langle\langle 1,0,0\rangle,42\rangle \tag{13}$$

Now, *NODE*$(\langle 1,0,0\rangle)$ is the final destination of the packet. Therefore, the message is sent via channel *out*. This creates the following event on the local output channel

$$out.\langle 1,0,0\rangle.42 \tag{14}$$

3.2. CSP mailer model

The model starts with constant definitions. There is only one constant n which governs the system setup. The number of *NODE* processes is 2^n. In the following equation we set n equal to 3 which results in a network with 8 *NODE* processes, as shown in Figure (3).

$$n = 3 \tag{15}$$

The next part of the CSP model is concerned with helper functions and their recursive implementations. The function adj(l), defined in Equation 16, returns a set. This set contains the IDs of all nodes to which the node with ID l is adjacent. A node is adjacent if the ID differs in only one digit.

$$\text{adj}(l) = \text{adj\_r}(l, \{\}, 0) \tag{16}$$

Equation 17 states adj_r(l, res, i) as the recursive implementation of adj(l).

$$\begin{array}{l}\text{adj\_r}(l, res, i) = \\ \left(\begin{array}{l}\textbf{if } i = \#l \textbf{ then} \\ \quad res \\ \textbf{else} \\ \quad \text{adj\_r} \\ \quad\quad \left(\begin{array}{l} l, \\ \text{vset\_m}(i, \text{b\_not}(\text{vget\_m}(i,l)), l) \cup res, \\ i+1 \end{array}\right)\end{array}\right)\end{array} \tag{17}$$

where $\#$ returns the dimension of the vector, vget_m(i,v) returns the ith entry of vector v, b_not(bit) returns 1 if $bit = 0$

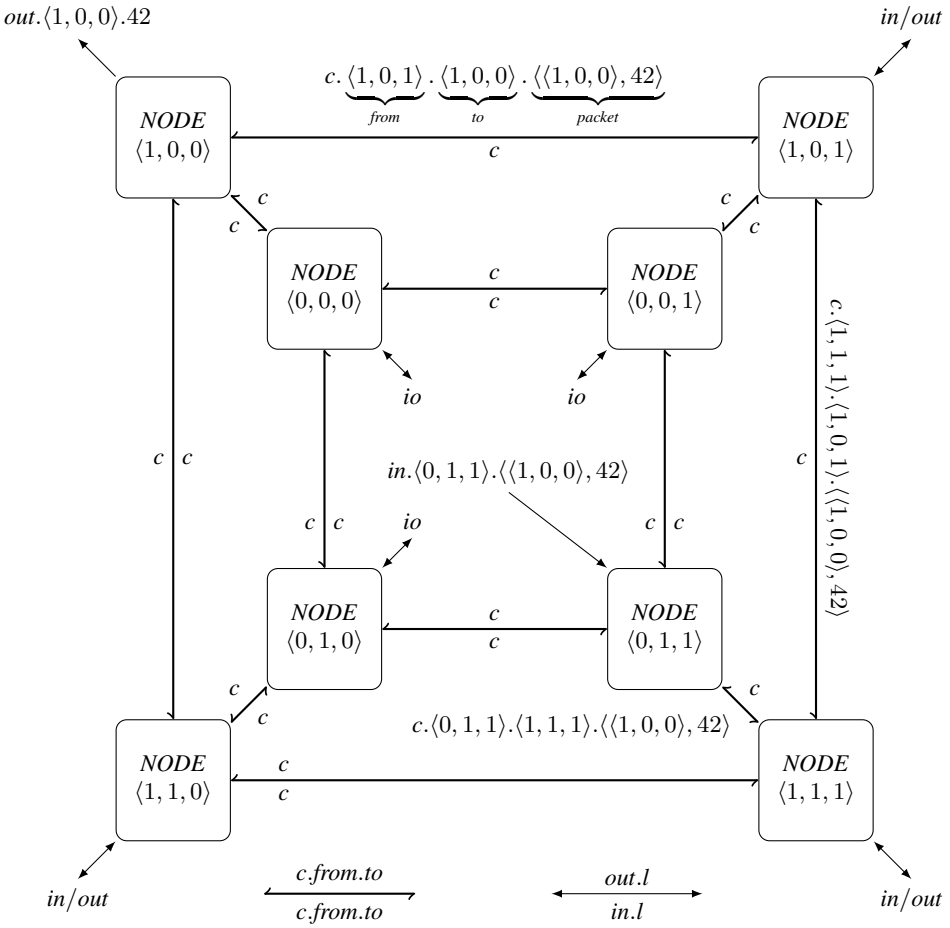

Fig. (3). *MAILER* process network for $n = 3$

else it returns 0, finally the function vset_m(i, val, v) changes the ith entry in vector v to *val*.

The function next(l, d), defined in Equation 18, returns the ID of the mailer node to which the message must be routed. The second part, defined in Equation 19, states the recursive definition of this functionality.

$$\text{next}(l, d) = \text{next\_r}(l, d, 0) \tag{18}$$

$$\text{next\_r}(l, d, i) = \left(\begin{array}{l} \textbf{if } \text{vget\_m}(i, l) \neq \text{vget\_m}(i, d) \textbf{ then} \\ \quad \text{vset\_m}(i, \text{vget\_m}(i, d), l) \\ \textbf{else} \\ \quad \text{next\_r}(l, d, i + 1) \end{array} \right) \tag{19}$$

A packet has the format $\langle address(d), message(m) \rangle$. The following two functions extract *message* and *address* respectively.

$$\text{get\_message}(\langle d, m \rangle) = m \tag{20}$$

$$\text{get\_address}(\langle d, m \rangle) = d \tag{21}$$

Equation 22 defines *all_nodes* as a set which contains all node IDs.

$$all\_nodes = \{\text{int2vec}(x, n) \mid x \in \{0..\text{pow}(2, n) - 1\}\} \tag{22}$$

where pow(x, y) returns x^y and int2vec(x, n) returns an n dimensional vector where the entries are the binary representation of the integer x. The constant *data* defines a set with all events that can be transferred over the routing channels.

$$data = \left\{ \begin{array}{l} \langle \text{int2vec}(x, n), y \rangle \\ \mid x \in \{0..\text{pow}(2, n) - 1\}, y \in \{1\} \end{array} \right\} \tag{23}$$

Now, we are ready to start with the definition of the atomic *NODE* processes. The functionality is so complex that the process is broken into four processes.

The first of these processes is $NODE(l)$. This process accepts a packet from the routing channels $c.i.l$ where $i \in \text{adj}(l)$ or a message from the input channel $in.l$ before it behaves like *CHECK*.

$$\begin{array}{rl} NODE(l) = & \Box_{i \in \text{adj}(l)}\, c.i.l?m \rightarrow CHECK(l, m) \\ & \Box\ in.l?m \rightarrow CHECK(l, m) \end{array} \tag{24}$$

where l is the node index and $c.i.l?m$ indicates that the process inputs the message m from the channel $c.i.l$.

The second process $CHECK(l,m)$ checks whether or not it is the destination of the packet. If it is the destination, then the process behaves like *OUTPUT* else like *ROUTE*.

$$CHECK(l,m) = \left(\begin{array}{l} \textbf{if } \text{get\_address}(m) = l \textbf{ then} \\ \quad OUTPUT(l,m) \\ \textbf{else} \\ \quad ROUTE(l,m) \end{array} \right) \quad (25)$$

where l is again the node index and m is the data packet.

The *ROUTE* process sends the packet to the next *NODE* process on the shortest path to the destination. After the message is sent out, the process behaves like *NODE* again.

$$\begin{array}{l} ROUTE(l,m) = \\ \quad c.l.\text{next}(l, \text{get\_address}(m))!m \rightarrow NODE(l) \end{array} \quad (26)$$

where $c.l.\text{next}(l, \text{get\_address}(m))!m$ outputs the message m over the channel $c.l.x$ and x is the next *NODE* process to which the message is routed.

The $OUTPUT(l,m)$ sends the message part of the packet over $out.l$ before it behaves like *NODE* again.

$$OUTPUT(l,m) = out.l!m \rightarrow NODE(l) \quad (27)$$

Equation 28 defines $\alpha NODE$ as the set of all events in which a particular process $NODE(i)$ can engage. In CSP this set is known as the alphabet.

$$\alpha NODE(i) = \{| \; in.i, out.i, c.x.i, c.i.x \; | \; x \in \text{adj}(i) \; |\} \quad (28)$$

The *MAILER* process is the parallel combination of all the 2^n individual *NODE* processes.

$$MAILER = \|_{i \in all\_nodes}^{\alpha NODE(i)} NODE(i) \quad (29)$$

the alphabetised parallel operator defines that a particular $NODE(i)$ synchronises with all events of its alphabet.

3.3. Stability considerations

We use the FDR tool to find the deadlock condition. Equation 30 instructs the tool to perform a deadlock analysis.

$$\textbf{assert } MAILER \text{ :[deadlock free [F]]} \quad (30)$$

Figure (6) on Page 68 shows that there is at least one deadlock condition in the system.

We use the debugging capabilities of the FDR tool to identify the deadlock state. Figure (4) shows the *MAILER* process network diagram generated by the FDR tool. In the FDR GUI this figure represents the *MAILER* process in the deadlock state. That means it is possible to study the set of events which a particular process accepts when the complete system is in the deadlock state. Similarly, it is possible to determine the set of events which a particular process refuses in the deadlock state. Furthermore, it is possible to see the trace of each process. The trace of the *MAILER* process constitutes the sequence of events which lead to deadlock. Equation 31 shows a possible deadlock trace of the *MAILER*. From this trace it is clear that this particular deadlock occurs when each node process receives a packet from the environment via the *in* channel. Effectively, this blocks the complete network, because all *NODE* processes try to send a packet over the routing network and no *NODE* process is able to receive.

$$\begin{array}{l} \langle in.\langle 1,1,1\rangle.\langle\langle 1,1,0\rangle,1\rangle, \\ in.\langle 0,0,0\rangle.\langle\langle 0,1,1\rangle,1\rangle, \\ in.\langle 0,1,1\rangle.\langle\langle 1,0,1\rangle,1\rangle, \\ in.\langle 1,0,0\rangle.\langle\langle 0,0,0\rangle,1\rangle, \\ in.\langle 1,0,1\rangle.\langle\langle 0,0,1\rangle,1\rangle, \\ in.\langle 0,1,0\rangle.\langle\langle 0,0,0\rangle,1\rangle, \\ in.\langle 1,1,0\rangle.\langle\langle 1,1,1\rangle,1\rangle, \\ in.\langle 0,0,1\rangle.\langle\langle 0,0,0\rangle,1\rangle\rangle \end{array} \quad (31)$$

3.4. *MAILER_ARBITER* CSP model

In this section we extend the *MAILER* functionality with the *ARBITER* process in order to avoid the deadlock. Figure (5) shows an *ARBITER* centric view of the *MAILER_ARBITER* system. The *MAILER* functionality is just represented by the *NODE* processes. As a matter of fact the *NODE* processes can be seen as the interface of the *MAILER*.

The system relies on the blocking capability of the $p2arb$ channels. Each producer $P(i)$ of a message tries to register with the *ARBITER* via the $p2arb$ channel. The *ARBITER* selects only one event from the connected $p2arb$ channels and blocks all other communication on these channels. After having accepted a message from a producer $P(i)$ the *ARBITER* expects a message from one consumer $C(i)$ via a $c2arb$ channel. The *ARBITER* does not know from which consumer $C(i)$ it receives the message: this is determined by the *MAILER* functionality. If a message, or in this case an event, was received over a $c2arb$ channel the *ARBITER* recurses and allows another producer $P(i)$ to register.

From this description it is clear that the *ARBITER* functionality ensures that only one message is processed by the *MAILER*. This effectively prevents the *MAILER* from deadlock.

3.5. CSP semantics

The functionality of the producer processes $P(i)$, provided in Equation 32, is straightforward. The $in\_ext.i$ constitutes the external interface of $P(i)$, via this interface the environment can send packages through the *MAILER* process network. First the process announces to the *ARBITER* that it wants to transfer a packet though the *MAILER*. If this request is granted, i.e. an event on the $p2arb.i$ channel occurs, the process is willing to input a packet from the $in\_ext.i$ channel. After that the process sends out this packet via the $in.i$ channel before it recurses. Note, the sequence of acquiring the permission from the *ARBITER* first before inputting the packet from the in_ext channel reduces the model complexity. Without loss of stability it would be possible to receive the packet first before permission is acquired. However, this method introduces a buffer which moderately increases the model complexity. But this moderate increase in model complexity nearly doubled the processing resource requirements of the FDR tool.

$$P(i) = p2arb!i \rightarrow in\_ext.i?x \rightarrow in.i!x \rightarrow P(i) \quad (32)$$

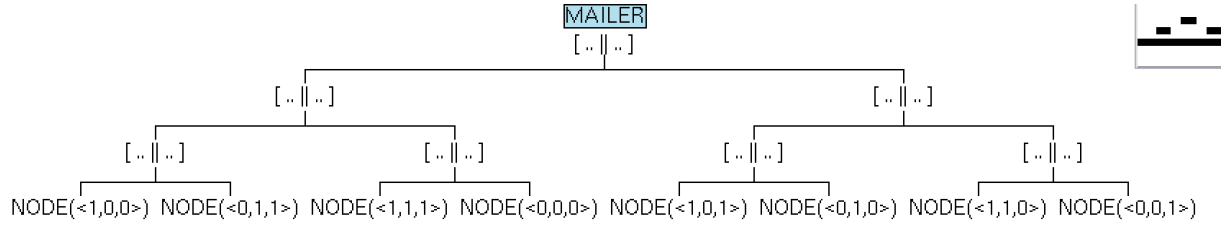

Fig. (4). *MAILER* process network generated by the FDR tool

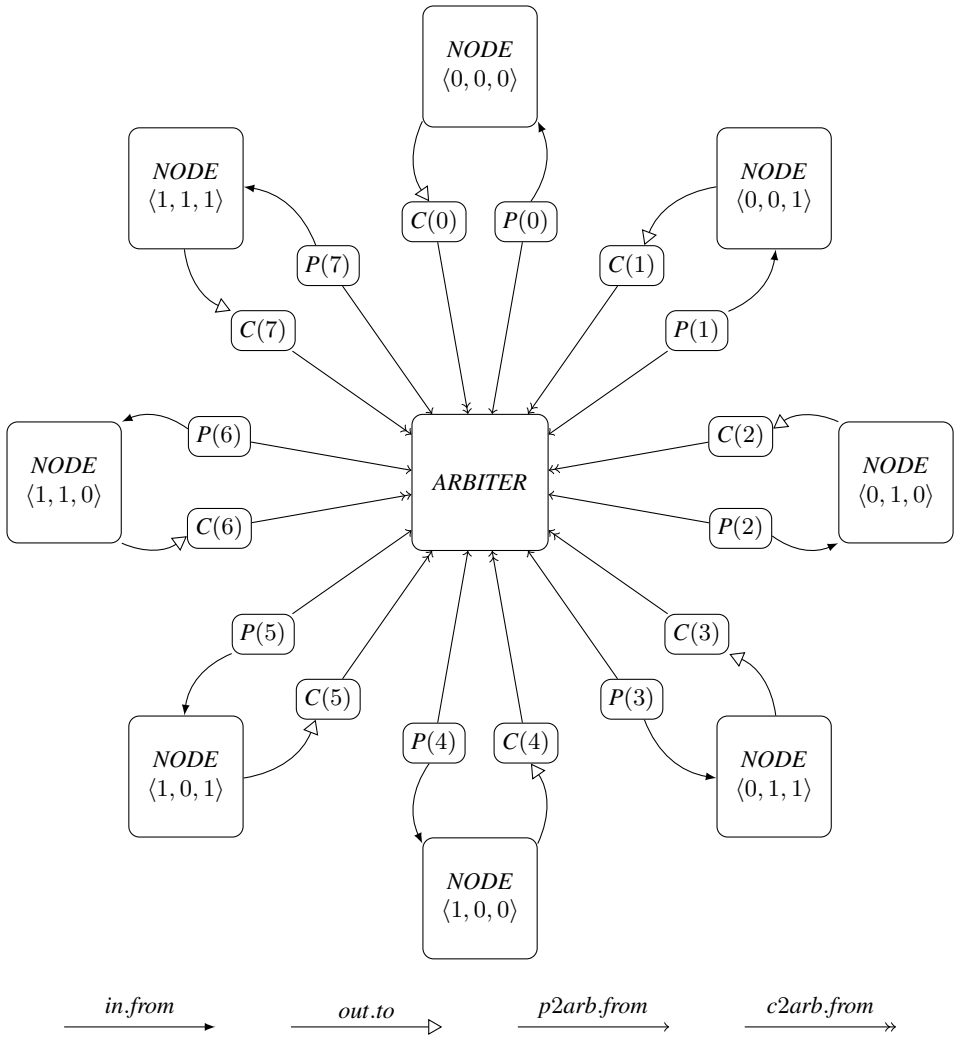

Fig. (5). Process network of *MAILER_ARBITER*

The consumer process $C(i)$, defined in Equation 33, consumes a message from the *out.i* channel and to announce the consumption via the *c2arb.i* channel.

$$C(i) = out.i?x \rightarrow c2arb!i \rightarrow C(i) \tag{33}$$

The *ARBITER* is the last atomic process of the current model. The process chooses one input event from a *p2arb* channel. After that it is only willing to accept an event from one *c2arb* channel. The reception of an event from a *c2arb* channel indicates that a packet was routed through and has left the *MAILER* process, therefore it is safe for the *ARBITER* to recurse and thus allow another message to occur on one of the *p2arb* channels.

$$\begin{aligned} &ARBITER = \\ &\quad \Box_{from \in all\_nodes} p2arb?from \\ &\quad \rightarrow \Box_{to \in all\_nodes} c2arb?to \rightarrow ARBITER \end{aligned} \tag{34}$$

All producer processes act independently from one another. Therefore, all producer processes $P(i)$ are interleaved to form the *PRODUCER* process.

$$PRODUCER = |||_{i \in all\_nodes} P(i) \tag{35}$$

Similarly, the consumer processes execute independently from one another. The *CONSUMER* process is formed by interleaving all consumer processes $C(i)$.

$$CONSUMER = |||_{i \in all\_nodes} C(i) \tag{36}$$

The *MAILER_PRODUCER* process is a helper construct which increases the readability of the model. The process combines *MAILER* and *PRODUCER* with an interface parallel. Both processes have to agree on all messages of the *in* channel.

$$MAILER\_PRODUCER = MAILER \underset{\{|in|\}}{\parallel} PRODUCER \tag{37}$$

The *MAILER_PRODUCER_CONSUMER* process is another helper construct which combines *MAILER_PRODUCER* and *CONSUMER* via interface parallel. Both processes[†] have to agree on all messages which are sent via the *out* channel.

$$\begin{aligned} &MAILER\_PRODUCER\_CONSUMER = \\ &\quad MAILER\_PRODUCER \parallel_{\{|out|\}} CONSUMER \end{aligned} \tag{38}$$

Finally, we are able to construct the *MAILER_ARBITER* process as the parallel combination of *MAILER_PRODUCER_CONSUMER* and *ARBITER*. These processes have to agree upon all messages sent via the channels *p2arb* and *c2arb*.

$$\begin{aligned} &MAILER\_ARBITER = \\ &\quad MAILER\_PRODUCER\_CONSUMER \\ &\quad \parallel_{\{|p2arb,c2arb|\}} \\ &\quad ARBITER \end{aligned} \tag{39}$$

[†]Both *MAILER_PRODUCER* and *CONSUMER* represent process networks, therefore to write 'both process networks' would also be correct.

3.6. FDR tests of the *MAILER_ARBITER*

This section describes two tests. First we prove that the *MAILER_ARBITER* process is deadlock free. In a second test we show that the *MAILER_ARBITER* process behaves according to specification. The first test is concerned with stability and the second is concerned with security and functionality.

The following equation[‡] instructs the FDR tool to carry out a deadlock analysis on the *MAILER_ARBITER* process.

$$\textbf{assert } MAILER\_ARBITER : [\text{ deadlock free [F] }] \tag{40}$$

This constitutes already the first test. Figure (6) shows that the test was successful, therefore we have proven that the CSP model of the *MAILER_ARBITER* process is deadlock free. This increases the confidence in the stability of the system.

The second test is concerned with security and functionality. This test requires a specification against which the *MAILER_ARBITER* can be tested. The specification defines all the required functionality and more importantly it does not exhibit undesired behaviour. One criterion for a good specification is that it should be different and if possible simpler than the process to test. These criteria are sometimes hard to fulfil.

The following equation defines the *SPEC* process which serves as specification for the *MAILER_ARBITER* process. Due to the fact that the *MAILER_ARBITER* is an implementation model which must comply with certain system aspects, the functionality can be modelled in a much simpler process. From the outside, all the *MAILER_ARBITER* process does is to receive a package via one of the *in_ext* channels and then the message emerges on the *out.i* channel, where *i* is the address of the packet. This leads to the following simple description of the specification process *SPEC*.

$$\begin{aligned} &SPEC = \\ &\quad \Box_{i \in all\_nodes} in\_ext.i?p \\ &\quad \rightarrow out.\text{get\_address}(p).\text{get\_message}(p) \\ &\quad \rightarrow SPEC \end{aligned} \tag{41}$$

The first test is concerned with security. To be specific, we test whether or not one process is confined to a subset of functionality of the second process. In CSP this is done with the trace refinement operation. The following equation instructs the FDR tool to verify that *MAILER_ARBITER* can only exhibit a subset of traces of the *SPEC* process.

$$\begin{aligned} \textbf{assert } SPEC \quad &\sqsubset_T \quad MAILER\_ARBITER \\ &\quad \setminus \{|\, c, p2arb, c2arb, in \,|\} \end{aligned} \tag{42}$$

where the hiding operator '\' prevents internal events from being detected by an external observer.

The second test establishes the functionality of the *MAILER_ARBITER* process. This is done via cross refinement, i.e. we ask whether or not the specification exhibits a subset of functionality of the system under test. The following equation instructs the FDR tool to check whether or not the *SPEC* process exhibits a subset of traces of the *MAILER_ARBITER* process.

$$\begin{aligned} &\textbf{assert } MAILER\_ARBITER \setminus \{|\, c, p2arb, c2arb, in \,|\} \\ &\quad \sqsubset_T SPEC \end{aligned} \tag{43}$$

[‡]Line of code

Both tests, stated in Equations 42 and 43 were successful, this leads to only one conclusion: *MAILER_ARBITER* and *SPEC* can exhibit the same traces.

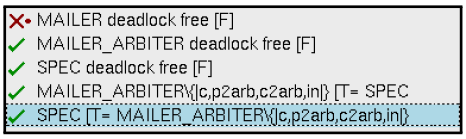

Fig. (6). Output of the model checker tool FDR

4. CONCLUSION

This paper discussed a pervasive design strategy for distributed health care systems. Such distributed health care systems suffer from the same problems as all distributed systems. Namely, it is difficult to establish security, stability and functionality of such systems. We addressed these problems by modelling distributed health care systems with the process algebra CSP. This leads to process oriented systems where everything is a process, from a single sensor to the complete health care system. The main idea behind CSP is that the individual processes communicate over well defined interfaces. This leads to systems which can be composed of simpler components. Furthermore, we could define system properties such as security, stability and functionality in a formal way. This leads to mechanised model checking, where specific system properties can be established.

In the practical section of this chapter we introduced a simple mailer system. The mailer represents a network of nodes which have the ability to send messages to one another. Such functionality is bread and butter for distributed health care systems. The first attempt to realise the design was straightforward: there is a meshed routing network between the individual nodes and each node acts as router, source and sink for the messages. The formal model of this system, to be specific the model checking of this formal model, unearthed a fundamental design flaw. The system deadlocks when all nodes try to send a message at the same time. Such a situation is unlikely, but it can happen. From the perspective of distributed health monitoring systems such a situation could indicate an emergency. And these are the situations when dependable systems are required. It is not acceptable that a system can be overwhelmed in case of an emergency. After the detection of the deadlock, the removal of the deadlock was trivial. The solution of having a central arbiter which ensures that only one message is in the routing network. In practical systems there might be other and better solutions.

We have abstracted distributed health care systems as a process network. This process network must comply with high demands on security, stability and functionality, because the underlying application is critical for human health. The pervasive design strategy helps us to achieve these high demands. Furthermore, this pervasive process oriented design strategy is the first step towards a deeper understanding of distributed health care systems. Based on this deeper understanding the next step towards decision making systems is possible. These decision making systems incorporate appropriate feedback structures which model artificial intelligence. They can also be understood with the pervasive design strategy. Therefore, the CSP based design strategy is truly pervasive: it opens the door for future projects on a secure, stable and functional basis.

REFERENCES

[1] Raghupathi W, Tan J. Strategic IT applications in health care. Commun ACM 2002;45(12):56–61.
[2] Greenes RA, H SE. Medical Informatics. An emerging academic discipline and institutional priority. JAMA. 1990;263:1114–1120.
[3] Consolvo S, Everitt K, Smith I, Landay JA. Design requirements for technologies that encourage physical activity. In: CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems. New York, NY, USA: ACM; 2006. p. 457–466.
[4] Toscos T, Faber A, An S, Gandhi MP. Chick clique: persuasive technology to motivate teenage girls to exercise. In: CHI '06: CHI '06 extended abstracts on Human factors in computing systems. New York, NY, USA: ACM; 2006. p. 1873–1878.
[5] Lee G, Tsai C, Griswold WG, Raab F, Patrick K. PmEB: a mobile phone application for monitoring caloric balance. In: CHI '06: CHI '06 extended abstracts on Human factors in computing systems. New York, NY, USA: ACM; 2006. p. 1013–1018.
[6] Silva JM, Zamarripa S, Moran EB, Tentori M, Galicia L. Promoting a healthy lifestyle through a virtual specialist solution. In: CHI '06: CHI '06 extended abstracts on Human factors in computing systems. New York, NY, USA: ACM; 2006. p. 1867–1872.
[7] Sohn M, Lee J. UP health: ubiquitously persuasive health promotion with an instant messaging system. In: CHI '07: CHI '07 extended abstracts on Human factors in computing systems. New York, NY, USA: ACM; 2007. p. 2663–2668.
[8] Tamura T, Kawarada A, Nambu M, Tsukada A, Sasaki K, Yamakoshi KI. E-Healthcare at an Experimental Welfare Techno House in Japan. The Open Medical Informatics Journal 2007;1:1–7.
[9] Bental D, Cawsey A. Personalized and adaptive systems for medical consumer applications. Commun ACM 2002;45(5):62–63.
[10] Kuhn KA, Giuse DA. From Hospital Information Systems to Health Information Systems. Method Inform Med 2001;4:275–287.
[11] Huston TL, Huston JL. Is telemedicine a practical reality? Commun ACM 2000;43(6):91–95.
[12] Fulcher J. The use of smart devices in eHealth. In: ISICT '03: Proceedings of the 1st international symposium on Information and communication technologies Trinity College Dublin; 2003. p. 27–32.
[13] Gelbord B, Roelofsen G. New surveillance techniques raise privacy concerns. Commun ACM 2002;45(11):23–24.
[14] Wilson EV. Asynchronous health care communication. Commun ACM 2003;46(6):79–84.
[15] Hoare CAR. Communicating sequential processes. Commun ACM 1978;21(8):666–677.
[16] Abdallah AE, Jones CB, Sanders JW, editors. Communicating Sequential Processes: The First 25 Years, Symposium on the Occasion of 25 Years of CSP, London, UK, July 7-8, 2004, Revised Invited Papers vol. 3525 of Lecture Notes in Computer Science Springer; 2005.
[17] Lin B, Vercauteren S. Hardware/Software Communication and System Integration for Embedded Architectures. Design Automation for Embedded Systems, KLUWER Journal 1997;2(8):359–382.

[18] Welch PH, Brown NC, Moores J, Chalmers K, Sputh B. Integrating and Extending JCSP. In: Communicating Process Architectures 2007; 2007. p. 349–369.
[19] Roscoe AW. CSP and determinism in security modelling. In: SP '95: Proceedings of the 1995 IEEE Symposium on Security and Privacy. Washington, DC, USA: IEEE Computer Society; 1995. p. 114.
[20] Kerridge J. Testing and Sampling Parallel Systems. In: Communicating Process Architectures 2007; 2007. p. 149–162.
[21] Welch PH. A Fast Resolution of Choice between Multiway Synchronisations. In: Barnes FRM, Kerridge JM, Welch PH, editors. Communicating Process Architectures 2006; 2006. p. –.
[22] Schneider S. Concurrent and Real-time Systems – The CSP Approach. 1st ed. 111 River Street, Hoboken, NJ 07030 United States of America: John Wiley; 2000.
[23] Failures-Divergence Refinement: FDR Manual. 26 Temple Street, Oxford OX4 1JS England; 1997. Available from: http://www.fsel.com/index.html.
[24] Roscoe AW. Theory and Practice of Concurrency. 1st ed. Upper Saddle River, New Jersey 07485 United States of America: Prentice Hall; 1997. Download: http://web.comlab.ox.ac.uk/oucl/work/bill.roscoe/publications/68b.pdf.
[25] Gao T, Massey T, Sarrafzadeh M, Selavo L, Welsh M. Participatory user centered design techniques for a large scale ad-hoc health information system. In: HealthNet '07: Proceedings of the 1st ACM SIGMOBILE international workshop on Systems and networking support for healthcare and assisted living environments. New York, NY, USA: ACM; 2007. p. 43–48.
[26] Ganesan D, Estrin D, Heidemann J. Dimensions: why do we need a new data handling architecture for sensor networks? SIGCOMM Comput Commun Rev 2003;33(1):143–148.

Received: March 03, 2008 Revised: March 27, 2008 Accepted: April 11, 2008