Dissertation

Ad Hoc Systems Management and Specification with Distributed Petri Nets.

Juan Sebastian Sosa Florez

December 11, 2021

This thesis is submitted in partial fulfillment of the requirements for a degree of Master in Systems and Computing Engineering (MISIS).

Thesis Committee:

Prof. Nicolás CARDozo (Promotor) Universidad de los Andes, Colombia
Prof. Paul LEGER Universidad Católica del Norte, Chile
Prof. Silvia TAKAHASHI Universidad de los Andes, Colombia
Ad Hoc Systems Management and Specification with Distributed Petri Nets.

© 2021 Juan Sebastian Sosa Florez

Systems and Computing Engineering Department
FLAG lab
Faculty of Engineering
Universidad de los Andes
Bogotá, Colombia

This manuscript has been set with the help of TeXShop and PDF\LaTeX{} (with Bm\LaTeX{} support). Bugs have been tracked down in the text and squashed thanks to *Bugs in Writing* by Dupré [11] and *Elements of Style* by Strunk and White [26].
DEDICATION
Abstract

Managing mobile ad hoc systems is a difficult task due to the high volatility of the systems’ topology. Ad hoc systems are commonly defined by means of their constituent entities and the relationships between such entities, however, a formal specification and run-time execution model is missing. The benefit of a formal specification is that it can enable reasoning about local and global system properties, for example, determining whether the system can reach a given state. We propose a Petri net-based specification and execution model to manage ad hoc distributed systems. Our model enables spontaneous communication between previously unknown system components. The model is locally equivalent to standard Petri nets, and hence could be used for the verification of properties for system snapshots static with respect to connections and disconnection, in which it is possible to analyze liveness, reachability, or conflicts. We validate the usability of our distributed ad hoc Petri net model by modeling distributable systems as described by existing distributed Petri nets approaches. Additionally, we demonstrate the applicability and usability of the proposed model in distributed ad hoc networks by implementing the communication behavior of two prototypical ad hoc network applications, disaster and crisis management, and VANETs, successfully validating the appropriate behavior of the system in each case.
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Connection situations for the ad hoc ping example</td>
<td>10</td>
</tr>
<tr>
<td>3.2</td>
<td>Distributed ping example model using DaPNs</td>
<td>14</td>
</tr>
<tr>
<td>3.3</td>
<td>High-level design of DaPN</td>
<td>21</td>
</tr>
<tr>
<td>3.4</td>
<td>Many-to-many communication between one ping and two pong nodes</td>
<td>23</td>
</tr>
<tr>
<td>4.1</td>
<td>Distributed Petri Net (DPN) example for mutual exclusion (Fig 8. in Badouel et al. [5])</td>
<td>26</td>
</tr>
<tr>
<td>4.2</td>
<td>Lock owner over time</td>
<td>27</td>
</tr>
<tr>
<td>4.3</td>
<td>Disaster and crisis manager model</td>
<td>28</td>
</tr>
<tr>
<td>4.4</td>
<td>Patient movement between the different nodes in the disaster and crisis manager case study</td>
<td>29</td>
</tr>
<tr>
<td>4.5</td>
<td>No. of nodes v. response time for patient throughput per node type</td>
<td>30</td>
</tr>
<tr>
<td>4.6</td>
<td>Model of a vehicular ad hoc network communication system</td>
<td>31</td>
</tr>
<tr>
<td>4.7</td>
<td>Information communication between vehicles and Road Side Units (RSUs) in a VANET</td>
<td>32</td>
</tr>
</tbody>
</table>
Ad hoc distributed systems are composed of multiple communicating nodes executing in different physical locations and at different moments in time. Due to distribution, computations may be delayed. Interactions and requests coming from multiple nodes at the same time need to be managed effectively. Petri Nets (PNs) have been successful in modeling and reasoning about concurrent and distributed systems \[ ? \[18\]. Existing proposals to model distributed systems with PNs \[29, 1, 5\] define strategies, conditions, and properties for classical distributed systems. Up until now, such proposals have not yet been explored to model ad hoc distributed systems.

In ad hoc distributed systems, the decoupling between computing nodes is even more latent, as these networks have a dynamic topology in which nodes can be continuously moving and, hence, connecting and disconnecting from the network \[28\]. Moreover, nodes in the network may not be known beforehand to other nodes, making their communication more difficult. In fact, in such systems it is not possible to apply existing distributability strategies \[29, 5\] since the composite nodes (and therefore the net topology) is unknown beforehand. Nonetheless, with the advent of the Internet of Things (IoT) \[22\], ad hoc systems are becoming ever more present and important.

### 1.1 Approach

In response to the phenomenon of growing ad hoc networks, this work presents a first approach to model, manage, and execute ad hoc distributed systems using PNs as an underlying formal model. We propose a PN-based model called DaPNs (Chapter 3), as an extension of standard PNs.
1 Introduction

Distributed systems are modeled using a DaPN for all of the system components deployed in each of the (independent) nodes in the network. Interactions between components are modeled by means of net composition, connecting transitions and places belonging to different nodes through a new type of arcs, remote arcs. The system dynamics are defined through an extension of the token game semantics, the distributed token game semantics, taking into account remote arcs, which allows transitions with remote arcs to interact with nets in other nodes, send messages in the style of the message passing communication model [9, 20]. The dynamic net behavior is specified by the casual transient connection [28] between DaPNs composing and splitting apart as nodes enter and leave a network.

1.2 Outline

In Chapter 2 we present multiple models of distribution and discuss their similarities to this work. Chapter 3 explains the Petri nets-based distributed model including the semantics, the behaviour and the implementation; all explained using a running example. Chapter 4 presents our validation using multiple instances of the model in the context of different application domains with real world applications. This chapter also discusses the results obtained. Finally, Chapter 5 posits the conclusion and explores avenues of future work.
Petri nets have been used to model distributability, compositionality, or adaptivity of (distributed) software systems. DaPNs, take inspiration, extend and refine some of the existing models, which we describe in the following. We discuss the relevant approaches in each of the aforementioned areas in perspective to our proposed model. Additionally, we highlight that none of the existing proposals addresses the problem of managing and executing distributed ad hoc systems, in which the complete system structure may be unknown beforehand, and is in continuous change as response to the (transient) composition of its nodes; our model is unique with respect to these characteristics.

2.1 Distributed and Distributable Models

We first discuss existing DPNs models. Similar to our approach, Adobbati et al. [1] present a definition of distributed net systems in which net places and transitions can be in different locations. In this work, the authors focus on modeling distributed systems with two locations (i.e., user and environment). In particular, their proposed DPN model is used to study liveness properties for the interactions between the two locations. However, the constraints imposed on the model are too restrictive to manage ad hoc systems, so the model is therefore not applicable to the domains in which multiple locations are needed. DaPNs enable the management of multiple locations, however liveness properties are not verified as in the DPN model. Such analysis could be transferred to DaPNs to reason about the liveness properties of the system.
2 Related Work

LSGA nets \([30, 31]\) are a class of DPNs \([29]\) in which all the components in the net are sequential locally, and communication between components in different locations is asynchronous. LSGA nets, based on component nets, are used to characterize distributable systems \([7]\). In particular, LSGA is used to characterize concurrency and conflict between actions in a software system. van Glabbeek et al. \([31]\) use this definition of DPNs to model distributable systems. That is, to reason about whether a system can be modeled with a DPN, in such cases the system could be realized as a distributed system. Similarly, Badouel et al. \([5]\) introduce the concept of distributable nets, based on separation theory, to model distributed systems through PN synthesis. The local properties of LSGA nets, and their asynchronous communication model between locations is similar to the proposal of DaPNs. However, in contrast to our approach, distributable nets do not actually distribute a PN, but describe a (possible) distribution of it. These systems state properties in which a given system, modeled as a PN, could be distributed. To describe such properties, the entire system must be known beforehand; which is not feasible for ad hoc networks due to the continuous mobility of their nodes. DaPNs can model ad hoc systems without knowing all nodes or locations, thanks to the dynamic discovery and composition of nets at run time (Section 3.3).

Reconfigurable nets \([3]\) enable modeling and verification of concurrent systems that change dynamically. Similar to our approach, the PN structure is not static, and as a consequence, the decidability of its properties is subject to the type of reconfiguration executed. Whenever reconfigurations are reversible, boundedness, reachability, and liveness are proven to be decidable \([4]\). The dynamics of DaPNs assure that reconfigurations are reversible, and so the results of reconfigurable nets could be used for the case of DaPNs. However it is unknown if the results hold in a distributed setting, and further research should follow for the new proposed application domain.

Mobile Nets and Dynamic Nets \([2]\) enable the possibility to add places and transitions to a given PN. In Dynamic Nets, new PN elements are generated as a consequence of transition firing. Nested Petri nets \([18]\) are used to model multi-agent distributed systems in a hierarchical structure. Similar to Dynamic Nets, Nested nets can too have complete PN structures flowing around due to transition firing. Both Dynamic and Nested nets allow the specification of different types of nets that effectively modify the behavior of the net. However, such behavior is anticipated taking
into account the transition firing rules, encoding the generation of net elements or synchronization between nets. This differs from our approach as in our case the PN structure changes through the spontaneous discovery or disconnection of distributed nodes. Therefore, we do not know beforehand the structure with which we are to modify the PN.

While different approaches to model distributed systems exist, after a revision of the related work, it is possible to see that the problem of ad hoc networks has special characteristics that have not been addressed up until now in existing PN-based modeling proposals, as for example, the upfront knowledge about the complete distributed system structure. With DaPNs it is possible to manage such cases, fill the missing gap for ad hoc networks.

2.2 Compositional Models

The DaPN model is based on the connection of distributed nodes to manage the global system behavior. Nodes’ behavior is specified by an extended PN structure. As a consequence, upon connection, the PNs for each component must compose to manage the complete system behavior.

The compositional Petri nets model proposes a modular definition for distributed systems using PNs [16]. This definition offers the semantics for the composition model of different PNs. In this model, two PNs compose through their interface places (i.e., places with the same label), defining behavioral patterns for input and output places. Interface places are used to model message-passing style communication between the different components. Such model of inter-component communication is similar to that proposed in Section 3.3. In DaPNs we choose to represent message-passing communication by means of far references [20] represented directly by remote arcs. Note that the definition of the semantics for the communication with remote arcs is equivalent to that of using interface places for one to one communication. However, our model also offers one-to-many and many-to-many communication using remote arcs.

Object nets [27] use the abstraction of nets-within-nets to represent agent-oriented systems. The specification of object nets uses the notion of nets-in-nets, in which net tokens are PNs themselves. Together with a set of rules (also represented as tokens), these nets can modify their structure dynamically by means of graph transformations [12] (i.e., rule-based modification). This specification could be used in modeling ad hoc distributed systems where distributed compo-
2 Related Work

Components are represented as token nets and their connection/disconnection triggers transformations in the net. The concept of nets-within-nets, and the general concept of graph transformations applied to PNs, is effective in managing systems composed in different locations, activating the correct behavior in each location (a net structure) according to the given rules. However, for the correct execution of these nets, it is necessary to have high-level knowledge of the interaction between locations. That is, knowledge of the nets to be active in each location, and the interaction rules between locations. In DaPNs, we abstract from any prior knowledge about the structure or interaction between locations, as, in the case of ad hoc systems, these are not fully known. Whenever, a node appears in a network, it tries to connect to other nodes in the network through the connection of remote interface places and remote arcs $f_r$. While the connection between remote nodes follows the same composition model, it is unknown, at run time, when and to which other nodes a particular node connects. The abstractions introduced with DaPNs address such challenge.

High-level nets [14] have a similar behavior as object nets seen as a nets-within-nets approach. However, these models also refer to Coloured Petri nets (CPN) [15] taking into account their modular and hierarchical characteristics. CPN introduce modules and interfaces as a way to organize large PN models. Modules allow us to organize PNs in small constituent components. Interfaces are defined as PN places used to compose different modules whenever their interfaces coincide. This definition of interfaces matches that of the compositional Petri nets model. As in that model, the use of modules and interfaces proposed in CPN is closely related with DaPNs. However, our model differs with this proposal in that it allows the remote communication with modules (of the same type) in multiple locations simultaneously.

2.3 Adaptive Models

Adaptive systems modify their structure or behavior in response to situations from their surrounding execution environment at run time. DaPNs recompose dynamically as the network changes. Nodes joining or leaving the network change the available components in the system as they connect or disconnect to the other system components (i.e., the DaPN). Therefore, the recomposition of DaPNs could be seen as the consequence of an adaptive system.
Adaptive Petri Nets [19] use the current state of a subset of places, known as *configurations points*, and a helper function to disable/enable transitions in a PN; effectively adapting the net behavior dynamically. However, this model presents a static PN structure. Furthermore, the dynamicity of behavior is based on a predefined set of rules arbitrating the current PN’s state. These two characteristics render this approach inapplicable in the ad hoc environment set for DaPNs. First, the structure of the net cannot be static, as nodes enter and leave the network unannounced. A static net structure would imply that the complete network topology is known beforehand. Similarly, the adaptation rules depend on a specific predefined PN state. Therefore, the possible net state would have to known beforehand. Both of these restrictions are unfeasible in the environments targeted by DaPNs, consequently, the adaptive Petri nets model is not applicable.

Similarly, the Self-Evolving Petri Nets [8] model is used to modify the structure of a given PN. To do so, the model defines different (static) rules specifying which PN elements to add or delete, based on its current state. As in the previous model, the adaptation of the net depends on evolution rules pre-defined according to specific PN states. In order to define the rules, the states and all elements to be added need to be known beforehand.

Distributed Context Petri Nets (DCoPNs) [13] use the distribution model of DaPNs to model and manage adaptive systems as realized by Context-oriented Programming [24]. In this model, a PN manages dependencies between behavioral adaptations defined in remote nodes through RemoteArcs and ServiceNodes, as discussed in our model.
3.1 Preliminaries

This section presents the model requirements for DaPNs, as well as the basic concepts used in the model definition.

As a motivation example, we take an ad hoc ping system. The system consists of two node types, a ping node and a pong node, that exchange messages with each other whenever they are in communication range (i.e., in the same network location).

In the initial system state, both node types are disconnected. Upon connection (i.e., both nodes join the same network) the ping node starts the communication by sending a message to the pong node (Figure 3.1a). In response, the pong node sends a message to the ping node, which reacts in a similar way, by returning the message to the pong node. This interaction continues so long the nodes remain connected.

Two situations can unfold as the system executes. First, the connection between the ping and pong nodes can break (Figure 3.1b). In such a case message exchange is stopped. The node in charge of responding with a message must remain idle until the connection resumes. Once the connection resumes, the system behavior should continue as if no interruption would have happened. Second, new nodes of either type can join the network unannounced. As a consequence they should connect with the respective node of the other type. For example, consider a second pong node connecting to the network (Figure 3.1c). The ping node now connects to both pong nodes, sending a message to both nodes whenever it responds to the original message, in turn
receiving responses from both nodes. If now a second ping node joins the network, it connects to both instances of pong, starting a message interaction with both nodes (Figure 3.1d).

3.1.1 Ad hoc Distributed Systems

Distributed systems are defined by a set of nodes \( \mathcal{N} = \{ N_1, \ldots, N_m \} \), each containing a set of software components. Nodes can reside in different physical locations delimited by their local network. Additionally, nodes can communicate with other nodes in the same location through message passing. Our model targets ad hoc distributed systems, therefore, we highlight two important properties of such systems [9, 28]. (1) Nodes are distributed in space and time. That is, nodes reside in different physical locations, and they may join and leave a location (i.e., network) unannounced, and (2) nodes have no previous knowledge of other nodes present in the network. Nodes are aware only of the interfaces they (currently) interact with (i.e., can send messages to or receive messages from). Such interfaces are called service names. Throughout the thesis we will use the sets \( I \) and \( I' \) to represent enumerable index sets.
Definition 3.1. Given a set of nodes $\mathcal{N}$ and a set of network locations $\mathcal{L} = \{l_1, \ldots, l_n\}$, we define $\lambda : \mathcal{L} \rightarrow (\mathcal{N}, m)$ with $m : \mathcal{N} \rightarrow \mathbb{N}$, as the assignment of nodes to a location, with possible repetition, such that $\lambda(l) = \{(\mathcal{N}_i, n_i) \mid i \in \mathcal{I} \text{ and } n_i \in \mathbb{N}\}$ defines a location as the multiset of all its nodes $\mathcal{N}_i$.

The system dynamics are given by the interaction of nodes as they join and leave a given location, as given in Definition 3.2. The concrete effect of nodes joining and leaving locations on DaPNs is discussed further in Section 3.3.1.

Definition 3.2. Given a location $l$ where $\lambda(l) = \{(\mathcal{N}_i, n_i) \mid i \in \mathcal{I} \text{ and } n_i \in \mathbb{N}\}$, we define nodes joining a location through the $\text{join}(N, l)$ predicate, such that $\lambda(l) = \{(\mathcal{N}_i, n_i) \mid i \in \mathcal{I} \text{ and } n_i \in \mathbb{N}\} \cup \{(N, 1)\}$. Similarly, nodes leave a location through the $\text{leave}(N, l)$ predicate, such that $\lambda(l) = \{(\mathcal{N}_i, n_i) \mid i \in \mathcal{I} \text{ and } n_i \in \mathbb{N}\} \setminus \{(N, 1)\}$.

In our ping example, Figure 3.1a represents a location with two nodes $\lambda(l) = \{(\text{ping}, 1), (\text{pong}, 1)\}$, Figure 3.1b represents the same location $l$ after the execution of the predicate $\text{leave}(\text{pong}, l)$ yielding a multiset $\lambda(l) = \{(\text{ping}, 1)\}$. Figure 3.1c is reached after executing the predicates $\text{join}(\text{pong}, l)$ and $\text{join}(\text{pong}, l)$ (effectively adding two instances of the pong node to the location) yielding $\lambda(l) = \{(\text{ping}, 1), (\text{pong}, 2)\}$. Based on this characterization of ad hoc distributed systems as locations containing node multisets that nodes can join/leave unannounced in response to network connectivity, we posit two requirements for an effective model for such systems.

*RQ*$_1$ The model should manage the spontaneous communication between previously unknown nodes in a given location.

*RQ*$_2$ The model should assure a reliable communication between nodes in the case of transient disconnections, to assure messages are not lost, and communication between nodes can resume.

### 3.1.2 Petri Nets

We now present the basic PN definitions used later to develop our DaPN model.
3 Modeling and Executing Distributed Ad hoc Systems with Petri Nets

**Definition 3.3.** A standard PN is defined by a tuple $\mathcal{P} = (P, T, f)$, where $P$ is a set of places, $T$ is a set of transitions, such that $P \cap T = \emptyset$, and $f : (P \times T) \cup (T \times P) \to \mathbb{N}$ is the flow function defining arcs between places and transitions, or transitions and places.

**Definition 3.4.** The pre- and post- sets of a PN element $x \in P$ are defined as:

- $\bullet x = \{ y \in T | f(y, x) > 0 \}$
- $x \bullet = \{ y \in T | f(x, y) > 0 \}$

Pre- and post- sets for transition elements (i.e., $x \in T$) are defined as in Definition 3.4, interchanging the roles of the places ($P$) and transitions ($T$) for $x$ and $y$.

**Definition 3.5.** A marking $m : P \to \mathbb{N}$ is a function assigning tokens to a place. $m(p) = n$ represents that there are $n$ tokens in place $p$.

**Definition 3.6.** Given a PN $\mathcal{P}$ with marking $m$, a transition $t \in T$, is enabled at $m$, under the standard PN semantics, denoted $m[t]_S$, if and only if $\forall p \in \bullet t$, $m(p) \geq f(p, t)$.

We use the subscript $S$ to represent the standard PN semantics and definitions. We omit the subscript whenever it is understood we are working with standard PNs.

**Definition 3.7.** Firing a transition $t$ enabled at marking $m$ ($m[t]$) leads to a new marking $m'$ where $\forall p \in P$, $m'(p) = m(p) - f(p, t) + f(t, p)$.

Markings are used to model the dynamics of PNs through the token game semantics. At each time-step, a transition may fire causing tokens to flow through the PN (by removing tokens from their pre-places and adding tokens to their post-places). A Marking represents the state (i.e., active places and their resources) of a PN at a given moment in time. The system dynamics is composed of a sequence of transition firings and the markings in between such firings.

### 3.2 Distributed Ad hoc Petri Nets

The definition of DaPNs builds on the definition of standard PNs, extending it with the possibility of having remote arcs. That is, arcs that communicate with PN elements in other nodes defined as remote services. We restrict remote arcs to flow from transitions to places only. The reason for
3.2 Distributed Ad hoc Petri Nets

this restriction is to map our model to the message passing communication style between remote nodes. Message sending starts by the event of transition firing. Therefore, firing a transition with remote arcs equates to sending a remote message between nodes. Additionally, this restriction assures that the local behavior of a node is consistent (with respect to its marking) regardless of the nodes connected to it. Definition 3.8 presents the definition for extended nets, in which remote arcs, \( f_r \), are present.

Remote nodes are connected following a zero-configuration network protocol in which a node can expose a service name (e.g., name) to be found by other nodes looking for the same service name. Service names are defined as any alpha-numeric string, and they do not need to be known beforehand by other nodes.

**Definition 3.8.** An extended net is defined as a tuple \( \mathcal{E} = (P_e \cup P, T, f, f_r) \) where \( P_e \) is the set of remote interface places, and \( P \) is the set of local places. Places in \( P_e \) are defined as tuples \((p, \text{name})\) where \( p \) is the place id, and \( \text{name} \) is the service name given to the remote interface. Places in \( P \) are standard \( PN \) places. \( T \) is the set of transitions with \((P_e \cup P) \cap T = \phi\). \( f \) is the \( PN \) flow function, and \( f_r : T \times \text{serviceName} \rightarrow P_e \cup \{\text{sentinel}\} \) is the flow function for remote arcs between different nodes.

Service names in remote interface places are used for communication discovery between extended nets in different nodes as we explain in Section 3.3. The sentinel resolution of remote arcs is used as a placeholder for the remote endpoint of the arc. Upon connection, the sentinel is replaced by the appropriate remote interface place (i.e., matching the service name).

**Definition 3.9.** Given an extended net \( \mathcal{E}_i \), we define the remote post-set of a transition \( t \in T_i \), as \( \bullet r = \{(id, \text{name}) \in P_e \mid f_r(t, \text{name}) = (id, \text{name})\} \) such that \( P_e \) is the set of remote interface places for an extended net \( \mathcal{E}_j \) with \( i \neq j \). Similarly, the remote pre-set of a place \((id, \text{name}) \in P_e\) is defined as \( \bullet r_p = \{t \in T_i \mid f_r(t, \text{name}) = (id, \text{name})\} \).

Note that in Definitions 3.8 and 3.9 \( f_r(t, \text{name}) \) is defined if and only if transition \( t \) is in node \( \mathcal{N}_i \), and place \((id, \text{name})\) is in node \( \mathcal{N}_j \) for \( i \neq j \). Additionally \( \mathcal{N}_i, \mathcal{N}_j \in \lambda(l) \) for a location \( l \) at a given moment in time.

**Definition 3.10.** \([\text{DaPN}]\) A Distributed Ad hoc Petri Net, \( \mathcal{D} \), is composed of a set of nodes
3 Modeling and Executing Distributed Ad hoc Systems with Petri Nets

\( \mathcal{N}_{i \in \mathcal{I}} \), each containing an extended net, such that at a given moment of time, for a location \( l \), \( \lambda(l) \subseteq \mathcal{N}_{i \in \mathcal{I}} \).

Each of the instances of Figure 3.1 shows a different composition of a DaPN. In Figure 3.1a we see a DaPN composed of two nodes, each represented a an independent DaPN. In Figure 3.1d we have a DaPN composed of four nodes, two instances of the \textit{ping} node and two instances of the \textit{pong} node.

As a DaPN \( \mathcal{D} \) is composed of different extended nets. We use the notation \( p \in \mathcal{D} \) to mean that place \( p \in \bigcup_{i \in \mathcal{I}} (P \cup P_e)_i \) in the extended net \( \mathcal{E}_i \) of node \( \mathcal{N}_i \) for \( i \in \mathcal{I} \). Similarly, \( t \in \mathcal{D} \) is used as shorthand for \( t \in \bigcup_{i \in \mathcal{I}} T_i \) where \( T_i \) is the set of transitions of \( \mathcal{E}_i \). Finally a marking \( m \) of \( \mathcal{D} \) is a token assignment function, such that \( \forall p \in \mathcal{D}, m(p) = n, \) for \( n \in \mathbb{N} \).

Figure 3.2 shows an example of a DaPN for a distributed \textit{ping} system in a given location. The DaPN is composed of two nodes. In the figure, the \textit{ping} component is in node \( \mathcal{N}_1 \) and the \textit{pong} component is in node \( \mathcal{N}_2 \). The extended nets are defined for the nodes are

\[
\mathcal{E}_1 = \{ (p_1, \text{ping}), \{t_{\text{ping}}\}, \{p_1, t_{\text{ping}}, 1\} \},
\]

\[
\{ \{(t_{\text{ping}}, \text{pong}, (p_2, \text{pong}))\} \}
\]

\[
\mathcal{E}_2 = \{ (p_2, \text{pong}), \{t_{\text{pong}}\}, \{p_2, t_{\text{pong}}, 1\} \},
\]

\[
\{ \{(t_{\text{pong}}, \text{ping}, (p_1, \text{ping}))\} \}
\]

In Figure 3.2 nodes are delimited by dashed lines, each containing an extended net.

Each of the extended nets defines a place which serves as both: a remote interface, and a processing unit for the component (e.g., \((p_1, \text{ping})\)). The extended nets additionally include a transition representing the component’s message passing interface (e.g., \(t_{\text{pong}}\)). Finally, the
extended nets have a remote arc crossing the node boundaries to other node, shown as thick (blue) arrows in the figure.

Without loss of generality, we assume that there is exactly one extended net per node. Having different extended nets in a node is equivalent to using a single extended net with multiple disconnected components.

3.3 Model Dynamics and Composition Semantics

The dynamicity of ad hoc distributed systems is due to the unannounced and continuous connection and disconnection of the nodes in the network. This causes the structure of a DaPN modeling such a system to continuously change. To manage transient connections and disconnections between nodes, our DaPN model is designed to arbitrate unannounced changes and interactions between the different network components.

3.3.1 Dynamic behavior of DaPNs

DaPNs dynamic behavior is defined by three key moments in nodes’ lifetime: connection, connected, and disconnection.

Whenever two nodes $N_1$ and $N_2$ join the same location $l$, they connect. Connection between nodes takes place by means of the instantiation of the remote arcs associating a node’s transitions with the other node’s remote interfaces ($P_e$) by means of their service names.

As previously explained, external places in an extended net define remote service interfaces. These interfaces are used as hooks to connect with other nodes by means of a service name (e.g., pong). A transition $t$ of an extended net may contain remote arcs. Such arcs are defined by specifying the service name, name, of the remote interface place $(p, \text{name})$ to which they are going to send remote messages, such that upon connection the arc’s end point is resolved to $f_r(t, \text{name}) = (p, \text{name})$.

**Definition 3.11.** [Dynamic connection] Take a location $l$ containing the DaPN $\mathcal{D}$ composed of $n$ nodes. Whenever a new node $N$ with DaPN $\mathcal{D}''$ joins the network by means of the $\text{join}(N, l)$
3 Modeling and Executing Distributed Ad hoc Systems with Petri Nets

The result of the join is a new DaPN $D'$ where: $\forall (t, \text{name})$ in which $t \in T_D$, and $\text{name}$ is a service name matching a service exposed by $\mathcal{N}$, such that $f_r(t, \text{name}) = (p, \text{name}), \forall (p, \text{name}) \in P_{r_D}$. Similarly, $\forall (t, \text{name})$ in which $t \in T_{D'}$, and $\text{name}$ is a service name matching a service exposed by one of the nodes defining $D$, $f_r(t, \text{name}) = (p, \text{name}), \forall (p, \text{name}) \in P_{r_D}$.

Definition 3.11 describes the way to resolve remote arcs dynamically, as the consequence of a node joining a new location. This can be seen in the ping system example as the remote interface of $N_2$ (i.e., the service name exposed by the node in its remote interface place $(p_2, \text{pong})$) corresponds to the service name defined for the remote arc of transition $t_{\text{ping}}$. Similarly, transition $t_{\text{pong}}$ in $N_2$ requires the remote interface $\text{ping}$ defined in the remote interface place $(p_1, \text{ping})$ of node $N_1$. The remote arcs in Figure 3.2 are resolved as $f_r(t_{\text{ping}}, \text{pong}) = (p_2, \text{pong})$ and $f_r(t_{\text{pong}}, \text{ping}) = (p_1, \text{ping})$. If a transition’s remote endpoint is not resolved, then we use sentinel instead, until a connection between the nodes is established.

Connection between nodes follows the zero-configuration networking model [25]. Two nodes are connected by resolving the endpoint of the remote arc $f_r(t, \text{name})$, whenever the service name $\text{name}$ corresponds in both nodes’ remote interfaces.

Once connected, remote messages can be sent between the source node (containing the transition $t$) and the receiver node (containing the remote interface place $(p, \text{name})$). In DaPNs, message sends are represented by means of remote transitions firing (cf. Definition 3.14).

Finally, if nodes are no longer in the same network, they disconnect and cannot exchange messages anymore. The result is the DaPN with remote arcs again resolving their remote endpoint to sentinel. As a consequence, upon disconnection, messages intended to the remote (disconnected) node are not sent, local messages are still forwarded to the appropriate PN places.

**Definition 3.12.** [Dynamic disconnection] Given a node $\mathcal{N}$ of a DaPN $D$ in location $l$, whenever $\text{leave}(\mathcal{N}, l)$ executes, then $\forall t \in D$, $f_r(t, \text{name}) = \text{sentinel}$ if $\text{name}$ corresponds to a service name of a remote interface of $\mathcal{N}$.

Note that in our definition, whenever a node joins a location, the result is a single DaPN structure. As a consequence, the join operation is agnostic to the multiplicity of nodes joining. That is, joining multiple nodes follows the exact same process, connecting the corresponding interfaces of all combinations of DaPNs in all joining nodes.
3.3 Model Dynamics and Composition Semantics

3.3.2 Composition semantics

The semantics of composing multiple nodes into a DaPN, defined next, is inspired by the composition semantics for PN components [16], adhering to the dynamic behavior of DaPNs described in the previous section.

Remember that remote arcs’ endpoints in an extended net resolve to a sentinel while the arc has not found a remote interface place to connect to. The sentinel is used as a placeholder representing the same identity as the remote interface place to which the net should send messages. While the sentinel is not a net object, it could be represented as a place. Therefore, whenever the extended nets are composed, they synchronize exactly at the sentinel/remote interface place.

Definition 3.13. Two extended nets

$E_1 = (P_e_1 \cup P_1, T_1, f_1, f_{r_1})$, and $E_2 = (P_e_2 \cup P_2, T_2, f_2, f_{r_2})$ are composable if $\exists (p, \text{name}) \in P_e_2$ and $t \in T_1$ such that a remote arc is defined between them $f_{r_1}(t, \text{name}) = \text{sentinel}$.

If two extended nets are composable by a remote arc between transition $t \in T_1$ and $(p, \text{name}) \in P_e_2$, the resulting extended net is $E_{1 \cup 2} = ((P_e_1 \cup P_e_2) \cup (P_1 \cup P_2), T_1 \cup T_2, f_1 \cup f_2, f_r)$ where

$$f_r(t, n) = \begin{cases} f_{r_1}(t, n) & \text{if } t \in T_1, (p, n) \in P_e_2 \\ f_{r_2}(t, n) & \text{if } t \in T_2, (p, n) \in P_e_1 \end{cases}$$

The composition of the extended nets generates a new DaPN composed of each extended net.

In our ping example, Figure 3.2, the nets for the ping and pong nodes are composable, as the ping extended net contains the transition $f_r(t_{\text{ping}}, \text{pong})$ and the pong extended net contains the external interface $(p_2, \text{pong})$. A similar situation occurs between the transition $t_{pong}$ and place $(p_1, \text{ping})$.

Similar to the join operation, the composition of multiple extended nets follows the process described in Definition 3.13 as the result of composing extended nets is again an extended net. Multiple compositions is managed two-by-two. Furthermore, note that if all the remote arcs of the extended nets in the composition resolve to sentinel the composition of the nets becomes the union of their places and transitions, ignoring the remote arcs.

DaPNs simulate message sending by firing transitions with remote arcs (i.e., arcs connecting a transition with a node from another location). This transitions correspond to the subset of $T$
for which $t\bullet$ is not empty (Definition 3.9). Therefore we extend the token-game semantics to take into account remote arcs.

**Definition 3.14.** Given a DaPN $D$, a transition $t \in D$ and a marking $m$, we say that $t$ is enabled at marking $m$ in the DaPN semantics, $m[t]_D$, if and only if $\forall p \in \bullet(t), m(p) \geq f(p, t)$.

The distributed token game semantics of a DaPN is based on a transition firing between two markings $m$ and $m'$ of $D$. Firing transition $t$ yields marking $m'$ ($m'$ is reachable from $m$ in the distributed semantics), $m[t]_D m'$, such that $\forall p \in D$:

$$
m'(p) = \begin{cases} 
m(p) + f(t, p) - f(p, t) & \text{if } p \in P \\
m(p) + 1 & \text{if } p \in P_e \\
& \land f_r(t, n) = (p, n)
\end{cases}
$$

Note, that the distributed token game semantics only allows sending one single message (point to point or broadcast) whenever a remote arc is used. That is, firing a transition $t$ with remote arc $f_r(t, \text{name}) = (p, \text{name})$ can only add one token to the remote interface places with service name $\text{name}$. In this semantics, the sent token is assumed to arrive to the remote arc endpoint, however, the message reception is not assured, as no confirmation message is issued by the remote endpoint. Furthermore, this implies that, within a node, the token game semantics corresponds to that of standard PNs. Theorem 3.1 shows the equivalence between the token game semantics of snapshots of DaPNs and standard PNs.

**Definition 3.15.** In DaPNs a snapshot of a net $D$ is defined as the static behavior of the net at a given moment in time. That is, connection or disconnection of nodes is disallowed. A snapshot of a DaPN $D$ corresponds to the composition of all extended nets $E_i \in \lambda(I)$, for all $i \in I$, defined as: $S = \langle \bigcup_{i \in I} (P_{e_i} \cup P_i) \cup \bigcup_{i \in I} T_i, f \rangle$, where, $f(p, t) = f_i(p, t)$ and

$$
f(t, p) = \begin{cases} 
f_i(t, p) & \text{if } p \in P_i \\
1 & \text{if } (p, n) \in P_{e_i} \land f_r(t, n) = (p, n) \\
0 & \text{otherwise}
\end{cases}
$$

**Theorem 3.1.** Given a DaPN $D$, and a snapshot $S$ (a static standard PN) of $D$, the distributed token game semantics of $S$ corresponds to the token game semantics in standard PNs.
Proof. Let $D$ be a DaPN composed of $m$ nodes $N_1, \ldots, N_m$, each with its corresponding extended net $E_i$, $1 \leq i \leq m$. Using the definition of a snapshot, the structure of $D$ is static at a particular moment of time. That is, no nodes join or leave the network ($\lambda$ does not change).

We need to prove that, for two given markings $m, m'$ at snapshot $S$, the following statements are true:

1. $\{t \in T \mid m(t)_D\} = \{t \in T \mid m(t)_S\}$

2. if $m(t)_D \Rightarrow m'(t)_S$ then $m(t)_D \Rightarrow m(t)_S$.

where $\cdot)_S$ represents the standard net semantics. The first statement follows trivially from Definitions 3.10. From Definition 3.10, we know that, for nodes already in the net, adding a new node cannot change the set of enabled transitions, as remote arcs only connect to places. Therefore, checking for enabled transitions disregards remote arcs. To prove the second statement, we first use the fact that we are taking into account only snapshots of $D$. That is, no connection or disconnection of nodes is allowed when calculating the set of enabled transitions, or firing a transition.

Suppose that $m'_D \neq m'_S$. That is, $\exists p \in D$ such that $m'_D(p) \neq m'_S(p)$, the marking of $p$ in the distributed semantics differs from the marking in the standard semantics.

Following Definition 3.14, the difference in the markings of places may exist only for those places that receive tokens from a remote arc. Therefore, $m'_S(p) = m_S(p) + f(t, p) - f(p, t)$, while $m'_D(p) = m_D(p) + f_r(t, p)$. Given that no disconnections can take place in the snapshot $S$, $f_r(t, p) = f(t, p)$ i.e., the token flow will not be affected by a disconnection. Moreover, from Definition 3.10, we know that, since $p \in t\bullet$, then $p \in N_i$ and $t \in N_j$ for $i \neq j$. As a consequence, $f(p, t) = 0$.

Then, $m'_D(p) = m'_S(p)$ as desired, and the distributed semantics is equivalent to the standard token game semantics.

Given that the semantics (i.e., dynamics) of our model is equivalent to the standard semantics in the static case (no connections or disconnections), we could reuse the same decision procedures used in standard Petri nets for the verification of properties of ad hoc distributed systems. This, however, can only occur for the extend of a connection, and the net would have to be reanalyzed.
upon every connection or disconnection. We further discuss the possible analysis directions for DaPNs as part of the future work in Chapter 5.

### 3.3.3 Running example

We now make the formalization of DaPNs concrete, and explain their dynamic behavior, using our ping ad hoc system.

The system is bootstrapped with the definition of each system node (i.e., component) as its own DaPN. Each of the individual extended nets $E_{\text{ping}}$ and $E_{\text{pong}}$ reside in independent nodes $\mathcal{N}_1$ and $\mathcal{N}_2$ respectively. Note that, as both DaPNs are disconnected (i.e., they are in different networks), the endpoints of the remote arcs $f_r(t_{\text{ping}}, \text{pong})$ and $f_r(t_{\text{pong}}, \text{ping})$ are sentinel services.

Suppose now that the nodes join the same location $l$ as they connect to the same network, using the predicates $\text{join}(\mathcal{N}_1, l)$ and $\text{join}(\mathcal{N}_2, l)$. As the DaPNs are now in the same location, they compose following Definition 3.13, yielding a unified extended net $E_{\text{ping} \cup \text{pong}} = \langle \phi \cup \{(p_1, \text{ping}), (p_2, \text{pong})\}, \{t_{\text{ping}}, t_{\text{pong}}\}, f_1 \cup f_2, f_r \rangle$ where the remote arc $f_r(t_{\text{ping}}, \text{pong})$ resolves to the service node place $(p_2, \text{pong})$ as endpoint, and $f_r(t_{\text{pong}}, \text{ping})$ resolves to the service node place $(p_1, \text{ping})$ as endpoint.

Once the nodes are connected, they can exchange messages. If, as in Figure 3.2, place $(p_1, \text{pong})$ is the only marked place in the initial marking $m_0$, then the only enabled transition is $t_{\text{ping}}$ ($m_0[t_{\text{ping}}]D$). Firing this transition yields a new marking $m_1$ where the only marked place (with one token) is $(p_2, \text{pong})$. The transition $t_{\text{pong}}$ now becomes enabled and can fire, sending a remote message to node $\mathcal{N}_1$, marking place $(p_1, \text{ping})$. While the two nodes are connected they will continue exchanging messages as in this example.

Suppose now that while $t_{\text{pong}}$ is enabled, the pong node leaves the network, $\text{leave}(\mathcal{N}_2, l)$. This causes the nets to disconnect and therefore divide the composed DaPN, into two different DaPNs, each with one of the original components for nodes $\mathcal{N}_1$ and $\mathcal{N}_2$. At this point it is no longer possible to communicate between the nodes. Communication will only resume if the nodes join the same location again; this time starting the communication by sending the first message from $\mathcal{N}_2$.
3.4 Implementation

DaPNs are implemented\(^1\) using the Go programming language\(^2\) to serve as a running model for ad hoc distributed systems. The DaPN model follows the structure of the ePetri Net Kernel \(^17\), with the addition of the communication components to enable node discovery \(^25\) and remote message passing communication \(^21\) \(^20\) between nodes, as shown in Figure 3.3.

![Figure 3.3: High-level design of DaPN](image)

The definition of DaPNs highlights two new structural entities from the standard PN definition, these are, RemoteArc and ServiceNode.

ServiceNodes are used to manage the decoupling in time and space of nodes. Nodes may define one or many services (i.e., through a service name) to communicate with other nodes. Remote interface places associate to a DaPN playing the role of Places in the net, and ServiceNodes in the node. ServiceNodes are defined as named servers that can receive messages from connected emis-saries (i.e., their remote input transitions). In DaPNs, places (or serviceNodes) are defined by specifying their id, service name, and marking. In the ping example (Figure 3.2), the \((p_1, ping)\) remote interface place is defined in Go as `pn.AddPlace(1, "ping", []pn.Token{pn.Token{1}})`, where `pn` is the PN instance, the first parameter 1 is the place id, "ping" is the service name of the remote interface, and the third parameter is the list of tokens in the place, defining its marking (in this example the list with a single token with identifier 1).

\(^1\)DaPNs implementation available at: [https://github.com/FLAGlab/DistributedPetriNets](https://github.com/FLAGlab/DistributedPetriNets)

\(^2\)https://golang.org
Remote interfaces are defined by means of remote arcs specifying remote interfaces and specializing remote message communication. RemoteArcs behave as clients in client-server systems. As a matter of fact, internally, a remote arc defines an HTTP connection to a ServiceNode via its service name. Once a connection is established between two nodes (i.e., the remote endpoint of the arc is resolved), the transition can send remote messages (i.e., send tokens) to the service node via an HTTP request. Remote arcs are defined in association with a transition via the transition id and the service name as: `pn.AddRemoteOutArc("t1", "pong")`, where the first parameter `t1` corresponds to the transition id, and the second parameter is the service name to which the client will connect and send messages (i.e., the remote endpoint). The weight of remote arcs is always set to 1.

As previously mentioned, we use zero configuration networking for service discovery. As nodes join a network, they query other nodes in the network for services that correspond to each of the service names defined in their remote interfaces (RemoteArcs) using a broadcast message. At the same time, all nodes already existing in the network query the new node to see if the new node defines a serviceName they require. Such discovery, and the subsequent communication process, is implemented based on the sleuth peer-to-peer library. If a connection between two nodes is possible, then the remote endpoint of the client defined by a serviceName in the RemoteArcs is resolved. Once a connection is established, the DaPN behaves as specified by the distributed token game semantics.

To support the many-to-many composition semantics of DaPNs the zero configuration communication model enables remote interfaces to interact with multiple nodes. If two nodes defining the same service name name are available in the network, and a node with a remote interface using name as service name joins the network, two client connections are generated for the remote interface, as show in Figure 3.4 for the ping example. Similarly, the two remote interfaces from the existing nodes connect to the new ServiceNode of the joining node, allowing it to receive messages from both nodes. In Figure 3.4 firing the tPong transition, will cause a message send to its two connected ServiceNodes (p2, pong) and (p3, pong). Firing each of the tPong transitions

---

3 Modeling and Executing Distributed Ad hoc Systems with Petri Nets

---

The HTTP connection is required by the sleuth discovery protocol, which we also use for message passing between nodes, however other communication protocols could be used, coupled with a different discovery protocol. [https://github.com/ursiform/sleuth](https://github.com/ursiform/sleuth)
3.4 Implementation

sends a message to \((p_1, \text{ping})\). Due to the token game semantics in DaPNs, only one message is sent at a time.

Finally, whenever a node leaves the network, the remote arcs connected to the node become unresolved (i.e., take the sentinel value), as well as all the remote arcs connecting the node to the DaPN. This process effectively divides the composed net into a component consisting of the leaving node, and a component with the remaining of the DaPN.

Figure 3.4: Many-to-many communication between one ping and two pong nodes
Validation

This section shows the feasibility of DaPNs in terms of their usability and genericity for modeling and executing ad hoc distributed systems. First, we show the compatibility of our model with existing proposals of DPNs (cf. Chapter 2). Second, we use DaPNs to model and execute distributed applications from different application domains in ad hoc networks.

For each of the case studies in the validation, we evaluate the response of the model to connect and communicate with unknown nodes ($RQ_1$), and the resilience to transient disconnections, in both the percentage of lost messages due to disconnections, and the continuation of communication as nodes reconnect ($RQ_2$).

All experiments were run using three physical devices, to model all nodes in the experiments. A core i5 with 8Gb of RAM Ubuntu machine, a core i7 with 16Gb of RAM Ubuntu machine, and a core i7 with 16Gb of RAM macOS 11. All experiments were run on a Docker container and Kubernetes as orchestrator with golang version 1.16, and our own fork of sleuth (v1.0.2) zero-config communication library. Finally our experiments use a LAN to connect the different node types. All the evaluation results for our experiments are available online in the project’s repository.

1 https://github.com/FLAGlab/sleuth
2 https://github.com/FLAGlab/DistributedPetriNets/tree/master/evaluation_results/2020-06-30
4.1 Distributed Petri Nets

4.1.1 Mutex.

As a first example, we implement the distributable mutual exclusion example taken from Badouel et al. [5]. In the distributed implementation of mutual exclusion, $M$ different (distributed) processes communicate via asynchronous message passing, assuring access to the critical section is pairwise disjoint for all processes. Figure 4.1 shows the two-processes example consisting of three nodes: a node for each of the processes ($P_1$ and $P_2$), and a node for the Mutex.

Each process defines a remote interface place to receive messages whenever they are granted access to the mutex, $(grant_1, grant_1)$ for process $P_1$, and $(grant_2, grant_2)$ for process $P_2$. The mutex node contains three different remote interface places, two places are dedicated to receive requests from the processes (one place for each process), namely, $(req_1, req_1)$ and $(req_2, req_2)$. The third place $(exit, exit)$ is a common interface for all processes to communicate the process is leaving the mutex. As a process, say process $P_1$, and the mutex discover each other, the two remote arcs $f_r(r_1, req_1)$ and $f_r(x_1, exit)$ from the process are resolved to the mutex’s places defining the corresponding service name. Conversely, the mutex’s remote arc $f_r(\tau_1, grant_1)$ is resolved to the corresponding service name in process $P_1$.

Figure 4.2 shows the evaluation of resiliency of the DaPN to manage the lock ownership, by making sure that the lock is not lost over time, and the processes do not access the lock simultaneously. The labels on the $y-axis$ in the graph correspond to the nodes: Mutex (1), $P_1$ (2), and $P_2$ (3) (each running on an independent physical device). The line in the figure represents the time steps in which each of the nodes had the lock (if the token is in the Mutex, then none of
4.2 Modeling and Executing Ad hoc Systems

In this section we use our DaPN to model and implement common use cases and applications of ad hoc networks.

4.2.1 Disaster and crisis management

The disaster and crisis management scenario represents one of the prototypical examples of ad hoc networks due to the spontaneous interaction between the system entities. In this scenario, several first response teams must coordinate their operation to successfully respond to a disaster. Disaster sites are divided in different remote areas to which responder teams may have access, to collaborate in different tasks. Firefighters and police officers usually have access to the incident area (IA) — that is, the place in which the disaster happened, and interact in rescue missions depending on gathered sensor information. In the casualty clearing stations area (CCSA), patients wait for their triage from paramedics, and their transport to the corresponding hospital. The transport area is the area in which patients are transported into ambulances (A), by hos-
4 Validation

pital personnel. Finally, the coordination of the complete operation takes place in the technical operation command area (TCO) (not shown in Figure 4.3), where the different interacting teams are assigned their locations and tasks [23].

Figure 4.3 shows an example instantiation of the ad hoc network for a disaster situation modeled using DaPNs. In this case, police officers ($P_1, \ldots, P_N$) and firefighters ($F_1, \ldots, F_N$) are available on site, and are given a rescue task whenever they are in proximity of the IA. The rescue tasks for each patient in the IA (represented as tokens) are assigned at random to an available rescue team member. For the disaster, the TCO established multiple CCSAs ($CCSA_1, \ldots, CCSA_M$) for the patients to wait for their triage. Patients, rescued by either police officers or firefighters, can go to any of the CCSAs available (the ones they connect to). Finally, after triage, the patients are assigned to any of the available ambulances ($A_1, \ldots, A_K$) in the transport area.

![Figure 4.3: Disaster and crisis manager model](image)

We evaluate the disaster management using the following scenario setting. All actors of the same type in the system (i.e., IA, $P_i$, $F_j$, $CCSA_m$, and $A_k$) are deployed in a single node independent from all others. We define different networks containing the nodes for each of rescue area types IA, CCSA, and Hospital. The $P_i$, $F_j$, and $A_k$ nodes join and leave these networks as they move patients around, taking them from the IA to an available CCSA, and then to a hospital. Such movement between the areas causes connections and disconnections between the nodes. To validate the effectiveness of the communication between the nodes, all nodes are
initially disconnected. Nodes connect one at time when they come in proximity of each other. In our experiment we have one IA, one P, one F, two CCSA, and one A nodes. We first evaluate the effective communication between nodes as they move around in the disaster area (RQ₂). If communication is successful, all patients initially in the IA should be accounted for in the ambulance A at the end of the simulation.

In the first experiment, Figure 4.4 shows the communication between the different nodes over time. Each of the horizontal lines represents one of the nodes (from bottom to top these are: ia, police, fire, ccsa1, ccsa2, ambulance). The X-axis represents the time-steps of the simulation, and each of the graphs (a)-(e) represent one of the patients originally in the IA place (i.e., the tokens in the initial marking). The ticks in the graph show the timesteps the patient spends at each node. Initially, all patients are in the IA node. As time passes, one by one, the patients start to be rescued by either the police officer (patients/tokens 1, 2, 3, and 5) or the firefighter (patient/token 4), who move the patient to one of the CCSAs (patients 2, 3, 4 are taken to the ccsa1 node, and patients 1 and 5 to the ccsa2 node). Finally, all patients are effectively rescued, as they are transported by the ambulance.

In the second experiment, we increase the amount of nodes and their dynamicity in the system, to evaluate the model’s reaction to more nodes, as they join and leave the network. We also assess the performance of the system by measuring the average time patients spend in each of the transition areas. We implement two scenarios; in the first scenario, in Figure 4.5a we increase
the number of CCSAs to three and in the second scenario, in Figure 4.5b, to four CCSAs. The number of police officers and firefighters varies dynamically and unannounced between 2 and 5 each, adding a new responder in intervals of approximately 40s ($RQ_1$). Both scenarios have three ambulances to transport patients to hospitals.

Figure 4.5 shows the time patients spent at each area (i.e., node type). We omit the time of transporting patients from the IA to a CCSA by police officers or firefighters as such time is negligible. We first observe that increasing the number of CCSAs reduces the overall time it takes for the patients to be transported to the appropriate attending facility (the difference between Figure 4.5a and Figure 4.5b). While this is intuitive, it shows the benefits of our model being able to add additional nodes (resources) into the system as needed. We observe from our results (not shown in the figures) that the 2 police officers and 2 firefighters initially present in the system
transport 64% of the patients, leaving a scarce number of patients for the remaining 6 responders who incorporate dynamically. Nevertheless, adding additional responders dynamically enabled a faster transfer for the remaining 36% of the patients, again, highlighting the importance of our model enabling dynamic incorporation of nodes.

4.2.2 Vehicular Ad hoc networks (VANETs)

This scenario implements a VANET, consisting of multiple passing vehicles exchanging information (e.g., about road conditions or congestion) with RSUs. Roads are split into cells covered by different RSUs. Therefore, moving vehicles are continuously updating their connectivity status with the RSUs closest to them.

The communication cycle of vehicles we are implementing is based on the model presented by Baraka et al. [6], and is as follows. Vehicles sense data from a set of $M$ available ISM bands whenever they are in service channel slots. Initially, such information is directly stored in each of the Vehicle’s Spectrum Availability Databases (VSADBs). Each vehicle sends the information it has gathered in its VSADB to the RSU it is currently connected with. RSUs collect information in their Spectrum Availability Database (SADB), where it is aggregated and analyzed, to generate decisions that can be sent to (following) passing vehicles. Figure 4.6 shows the model of the system, in which $M$ sensors communicate information to a vehicle, which then sends such information to an RSU. Finally, the RSU communicates the aggregated received information with passing vehicles.

![Figure 4.6: Model of a vehicular ad hoc network communication system](image-url)
4 Validation

Similar to the crisis management case, we evaluate the communication of the VANET components as vehicles move along the road. The experiment setting consists of five different RSUs located along a city street, such that their connectivity areas do not intersect. Vehicles move along the street connecting to each of the RSUs, one at a time, sending the information they have collected, and receiving collected information from the RSU. Figure 4.7 shows the information flow through the different nodes in the network [IRA2]. For this experiment, we assume each vehicle and each RSU have information stored in their VSADB and SADB respectively. Such information is communicated between the different components as they move. We only allow for each of vehicles to exchange one message with each RSU.

![Diagram of information communication between vehicles and RSUs in a VANET](image)

Figure 4.7: Information communication between vehicles and RSUs in a VANET

In Figure 4.7, the X-axis represents system execution time-steps of the simulation, and the Y-axis represents the location of the corresponding entity. For example, at time-step 10, the information generated by dom’s car, is in RSU 2, as the car passed by that location (Figure 4.7a). As it is possible to see from the figure, tokens flow between RSUs as they are transported from one connectivity area to another, proving the ad hoc connection and information communication of DaPNs while they dynamically change locations.
4.3 Threats to Validity

In this section we discuss the threats to the validity of our evaluation.

*Threats to external validity* refer to the generalization of the results from our evaluation. Our evaluation explores the use of DaPNs to four different application domains with different behavioral properties, in which we are able to evidence the effectiveness of DaPNs in executing the modeled systems and managing the communication between remote nodes. However, we note the case studies used do not cover all aspects of all possible ad hoc distributed systems and the problems therein. Therefore, more extended evaluation exploring other types of ad hoc networks is desired.

*Threats to construct validity* refer to the cases in which the evaluation is not exhaustive or selective enough. In our case, the implementation of DaPNs follows the specification of the model. In particular, the composition of extended nets follows the process specified by our model managing the composition of nets as they connect to nets in other nodes. However, as many of the communication components are external, some of the internal details could not yet be specified and validated independently in our model. To fully specify our model taking into account such communication characteristics (e.g., remote messages or zero configuration) we could extend our formal model with concepts of existing specifications of distributed systems \[21, 10, 32\] (and vice versa).

*Threats to internal validity* refers to the conditions that may have an impact on the phenomena under test. In our experiments, these are represented in the instrumentation of the process to execute the tests, in which we used multiple VMs executing in three machines. Under this setting, the connection and disconnection of all the different combinations of nodes could not be tested in full, however, our validation demonstrates the partial-disconnection of the different node types in all the experiments, and their successful reconnection. Additionally, we managed disconnections by physically unplugging the node from the network; other more subtle types of disconnections could occur, and need to be tested.
Conclusion and Future Work

5.1 Conclusion

This thesis presents a Petri net-based model for the specification and execution of ad hoc distributed systems, called Distributed Ad hoc Petri Nets (DaPNs). DaPNs are a first approach to modeling and executing ad hoc systems by means of Petri nets. To define DaPNs, we extend the standard Petri nets definition with the concept of remote arcs — that is, allow communication between nodes across their boundaries, by means of remote message passing. Remote arcs are used to connect many such extended net models, whose composition (through ad hoc connections) constitutes a complete DaPN. To execute our model we extend the standard semantics to a distributed token game semantics, which takes into account remote arcs, as well as transient connections between nodes.

The validity of DaPNs is demonstrated from two perspectives. We first show the relation between the distributed semantics and the standard net semantics for the specific case of DaPN snapshots (static views of the net as if it was defined in a single node). This result is important to show that, at any given moment in time, a DaPN can be simulated with a standard Petri net. Second, we implement DaPN-based systems in four different application domains, in order to show the genericity of DaPNs in modeling ad hoc distributed systems. Furthermore, we execute the aforementioned systems, using multiple scenarios in each of the domains, demonstrating the capabilities of the model to seamlessly deal with transient network connections, as well as the reliability when passing information between remote nodes. We can therefore conclude that
5 Conclusion and Future Work

DaPNs are effective in modeling and executing ad hoc distributed systems.

5.2 Future Work

This thesis positions DaPNs as a viable model for the definition and execution of ad hoc distributed systems. However, the equivalence between our model and the standard Petri net model opens the opportunity to utilize the analysis and verification techniques of Petri nets to assure static properties (i.e., their snapshots) about ad hoc distributed systems.

A first approach would be to apply, as-is, existing Petri net verification techniques to ad hoc distributed systems through DaPNs. Such application could already yield valuable results to the distributed systems community. However, such analysis can be time-consuming and computationally heavy to use in a rapidly changing dynamic environment. DaPNs also open the possibility to explore incremental verification techniques for distributed systems, in which independent nodes assure a certain property (e.g., reachability) using existing Petri net verification techniques, as within a node the defined net corresponds exactly to a standard Petri net. Then upon connection, local results for each node could be reused to ease the verification of the larger system. A first step in this direction would be to apply reachability analysis of DaPNs for the verification of liveness properties in distributed systems.
The references are sorted alphabetically by first author.


Bibliography


Bibliography


