

Formal Specification and Validation of a Data-driven Software System for Fire Risk Prediction

Ruben D. Strand¹, Laure Petrucci² and Lars M. Kristensen³

¹*Department of Safety, Chemistry, and Biomedical laboratory sciences, Western Norway University of Applied Sciences, Haugesund, Norway*

²*LIPN, CNRS UMR 7030, Université Sorbonne Paris Nord, Villetaneuse, France*

³*Department of Computer science, Electrical engineering and Mathematical sciences, Western Norway University of Applied Sciences, Bergen, Norway*

Abstract

Long periods of dry and cold weather conditions significantly increase fire risks for wooden buildings. Recent advances in predictive fire risk models combined with publicly available cloud-based weather data services have enabled the development of smart software systems for location-oriented fire risk notification. We have developed a Coloured Petri Net (CPN) model specifying the software architecture of a microservice-based predictive fire risk notification system. The CPN model captures the set of micro-services provided via REST APIs and the interaction between the constituent services for location tracking and subscription, fire risk computation, and data harvesting. As part of the work, we present a general modelling approach and pattern for REST-based APIs. We apply simulation and state space exploration to validate and verify key behavioural properties of the predictive fire risk notification system.

Keywords

Coloured Petri nets, modelling, verification, microservices and REST APIs, software systems architecture, fire risk prediction

1. Introduction

The pervasive presence of cloud-based data services provides access to a wide range of data sources that enable the development of data-driven applications supporting decision making and control. Prominent examples of such data sources includes weather data measurements and weather forecasts which can be used by authorities in the context of early warning systems, including smart systems for fire risk predictions.


The winter period in certain regions of Norway is characterised by longer periods with dry and cold weather conditions which combined with the high density of wooden houses poses a significant threat to many cities in Norway. An example of this was the incident on January 18th 2014, when at that date, the largest fire in Norway since World War II developed in the village of Lærdalsøyri at night-time [1]. Forty buildings were lost, including four historic buildings, in a fire that only developed between houses [2].

It is generally known that the winter in cold climate regions brings along increased fire frequencies [1, 3]. This was originally identified by Pirsko and Fons [4] which suggested

PNSE'22, International Workshop on Petri Nets and Software Engineering, Bergen, Norway, 2022



© 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

ambient dew point during the winter as an explanation for the increased fire frequency in buildings. More recently, Log [5] proposed indoor relative humidity as a fire risk indicator and developed a mathematical model [6] that predicts indoor fuel moisture content (FMC) for a wooden house. The model correlates the FMC with the time to flash-over (TTF) [7], resulting in the possibility of indicating fire development. Combined with forecast weather data, the TTF can be predicted for the upcoming days, enabling proactive emergency planning. Further, by combining the TTF with the influence of wind, a combined fire risk can be expressed, as presented in [8]. The basic idea underlying the model of Log is to use outdoor temperature and relative humidity to estimate (compute) indoor relative humidity which in turn enables computation of the wooden fuel moisture content (FMC). In Norway, measured weather data and weather forecast data (including temperature and relative humidity) are available via cloud-based REST APIs of the Norwegian Meteorological Institute (MET) [9, 10].

The work presented in this paper is conducted in the context of the DYNAMIC research project [11] which has as a main objective to develop a cloud-based predictive fire risk indication system based upon the fire risk prediction models developed within the project. In earlier work, we have validated the predictive fire risk indication model [12] demonstrating that it can provide trustworthy fire risk indications, that a combination of measurements and forecast data can be used, and that the weather data available from MET [9, 10] can be used to obtain fire risk indications of the correct order of magnitude (minutes). Furthermore, in earlier work [12] we implemented a simple first version of the system, which served as a basis for the development of a prototype of the fire risk notification system based on a micro-service software architecture. This prototype was developed through an iterative process in parallel with developing a first version of the CPN model. However, the development of the CPN model was based on the prototype system, hence developed a bit behind the prototype. This resulted in a supportive development process, as the CPN modelling continuously revealed problems and identified possible pitfalls within the prototype. This assured a more robust and complete development of the system. The prototype system and the CPN model were developed by different people, which was considered to strengthen the development process. The prototype implementation was implemented using the Heroku [13] and Amazon EC2 cloud platforms and the Spring Boot framework [14] in combination with MongoDB noSQL databases to realise the micro-services. In addition, a web-based front-end application was prototyped using the React single-page web application framework, and a mobile front-end was prototyped using the Xamarin cross-platform application framework [15]. The current paper reports on a second version of the CPN model, which took place after the supportive development process. The updated CPN model will serve as basis for the final system implementation.

Contribution. The contribution of this paper is three-fold: Firstly, to take a step back from the initial prototype implementation and develop a formal specification of the software system architecture and the constituent micro-services ; Secondly, to develop some general modelling patterns for REST-based micro-services ; and thirdly to verify the system services and the behavioural interaction between the micro-services. A main benefit of the constructed CPN model is that it provides an implementation independent specification of the fire risk notification system and because of the initial prototyping we ensure the implementability of the system.

Related work. Several works in the recent literature have addressed formal modelling of (micro-) services. Most of these works do not benefit from the data handling offered by Coloured Petri Nets nor provide a structured hierarchical design.

In [16], services are modelled using Timed Petri Nets (TPNs), thus focusing on the process flow and discarding the data exchanges. An automatically generated TPN then allows for verifying properties of an input microservice specified in the CONDUCTOR orchestration language. A similar approach for self-adaptive systems using high-level Petri nets was conveyed in [17].

The Saga patterns are extended with concurrency features in [18] via workflow nets. They are further translated into reference nets embedded in the RENEW tool which provides simulation features. It also suffers from the restricted data representation and analysis capabilities. Services are organised using simple Petri net patterns in [19]. In [20], the authors propose a Coloured Petri net model for RESTful services, with a particular focus on composition issues. However, it does not exhibit a hierarchical model nor a general architecture for micro-services models.

A CPN-based case-study verification of a cloud-based information integration architecture was presented in [21]. Although modelling the different layers of the cloud-based architecture and verifying specific model properties, emphasis was not given to the communication, data exchange, between the layers. However, [22] presents a structuring of data and color sets which has similarities with the general communication between the REST-based micro-services presented in this paper. In their work, they considered the automatic generation of a CPN model for a distributed automation architecture.

Overview. The rest of this paper is organised as follows. Section 2 gives an overview of the fire risk notification system by presenting the top-level modules of the CPN model and exemplifying the basic interaction between the micro-services when the system is being executed. In Sections 3-5 we present how the three micro-services (business logic, fire risk computation, and data harvesting) have been modelled, as well as our general approach to modelling REST-based APIs. In Section 6 we formulate key behavioural properties for the fire risk notification system and explain how they have been validated and verified using simulation and state space exploration. Finally, in Section 7, we sum up conclusions and discuss future work.

2. Fire Risk Notification System and CPN Model Overview

The top-level CPN module of the system is presented in Figure 1. It consists of the substitution transitions *Client Applications*, *Fire Risk Notification System (FRNS)*, *External Weather Data Services* and associated socket places. The *Client Applications* represent any front-end service used by the clients to communicate with the FRNS, i.e. a web browser or mobile application. The FRNS represents the main fire risk notification system, consisting of 22 submodules. The substitution transition *External Weather Data Services* is representing weather data sources providing historical and forecast weather data.

The services communicate by producing and consuming tokens on the connected socket places. The tokens represent data transferred between the services. Each place is associated with a specific color set, allowing only a certain type or combination of data to be present. Considering a request from the *Client Applications*, a token is produced onto the place *Request*,

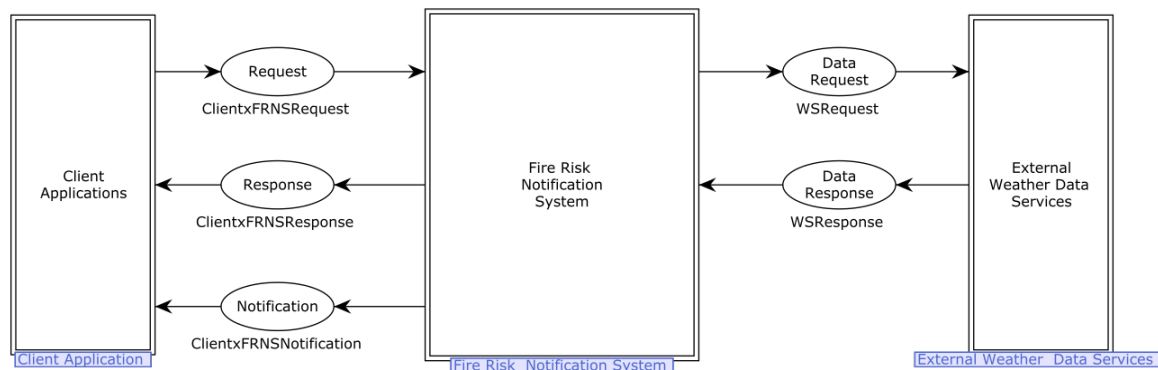


Figure 1: The top-level module of the CPN model, with the front-end client applications at the left, the main software component FRNS in the middle, and the external weather data services at the right.

connecting this application with the FRNS. In turn, the FRNS consumes the token, processes the request and responds by producing a token onto the place *Response*. The response token is then consumed by the *Client Applications*. The FRNS either responds to a request from the *Client Applications*, as described, or it notifies clients through scheduled services. This latter service results in one or more tokens being produced at the place *Notification*. The remaining two places in Figure 1 are the *Data Request* and *Data Response*. These are the places connecting the *Fire Risk Notification System* and the *External Weather Data Services*. Tokens are produced at these places whenever the FRNS needs historical and forecast weather data for fire risk computations.

Expanding on the system, Figure 2 shows the *Fire Risk Notification System* submodule. The module consists of three substitution transitions and associated socket places. The substitution transitions represent the individual services of *Business Logic Controller Service* (BLCS), *The Fire Risk Computation Service* (FRCS) and *The Data Harvesting Service* (DHS). The services are modelled as loosely coupled micro-services, communicating through defined REST API endpoints, in accordance with the previously developed prototype [23]. The BLCS and DHS micro-services are connected to the overall system in Figure 1 through the places with equal naming. Hence, the BLCS communicates with the *Client Applications* and the FRCS, while the DHS communicates with the external weather data sources and the FRCS. The services are only aware of the existence of directly connected services.

The BLCS is the middleware, separating the *Client Applications* and the FRCS. It processes and responds to requests from the clients by further requesting the FRCS. In addition to the RESTful services, it also handles the fixed interval process of notifying clients in case of high estimated fire risks. To receive a notification for a location, a client must be subscribed to that location. The subscription database, handled by the BLCS, keeps track of these subscriptions.

The FRCS is responsible for estimating the fire risks at considered (tracked) locations, which is primarily a scheduled service. Upon computing the fire risks, the FRCS requests weather data from the DHS. Then, if weather data is received, the FRCS predicts the upcoming fire risks and updates the local fire risk database. When requested by the BLCS, the FRCS can return estimated fire risks from the local database. Hence, the fire risk database is continuously updated.

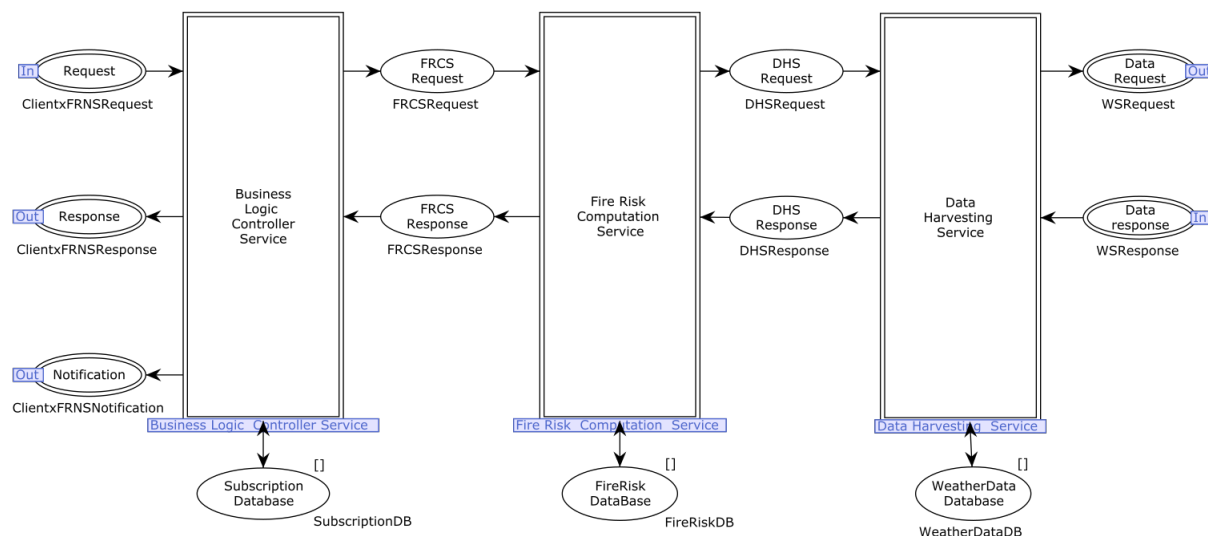


Figure 2: The FRNS submodule, with the BLCS at the left, the FRCS in the middle and the DHS at the right. The far left and right places correspond to the places in Figure 1.

The DHS receives weather data requests from the FRCS. It handles a weather database containing locations and associated weather data. The database is subject to scheduled updates, so when the FRCS requests weather data, the DHS can respond with recently fetched forecasts and measurements for all the locations within the database. Hence, the DHS functions similarly to the FRCS, by means of scheduled updates of local databases. The existence of locations is synchronised between the databases.

Figure 3 presents a sequence diagram for the main functions of the system. In general, clients may: (1) start or stop tracking of a new location; (2) subscribe or unsubscribe to existing locations to handle personal notifications; or (3) request fire risk for locations existing within the system. In order to provide fire risk notifications, the system needs to continuously monitor the current and future fire risk. This is achieved by a scheduled update, running every six hours. It starts by an update of the weather database at DHS, then the FRCS requests the recently fetched weather data, recomputes all fire risks and updates the fire risk database. Then, the BLCS requests all recently computed fire risks and determine which clients to notify, if any, based on certain fire risk criteria and the subscriptions within the subscription database.

Abstraction of data. Abstractions are made to the model to arrive at an appropriate level of detail. Although some details in data are useful for modelling in simulation purposes, they are not necessarily relevant when doing verification. The included data need only to correspond with the modelling objective. Further, abstraction of unnecessary data is important to reduce the size of the state space, ensuring a finite space and allowing a more exhaustive verification.

The abstractions made to the CPN model primarily relate to the payload of the messages. When the different services communicate, the purpose of the communication is more important

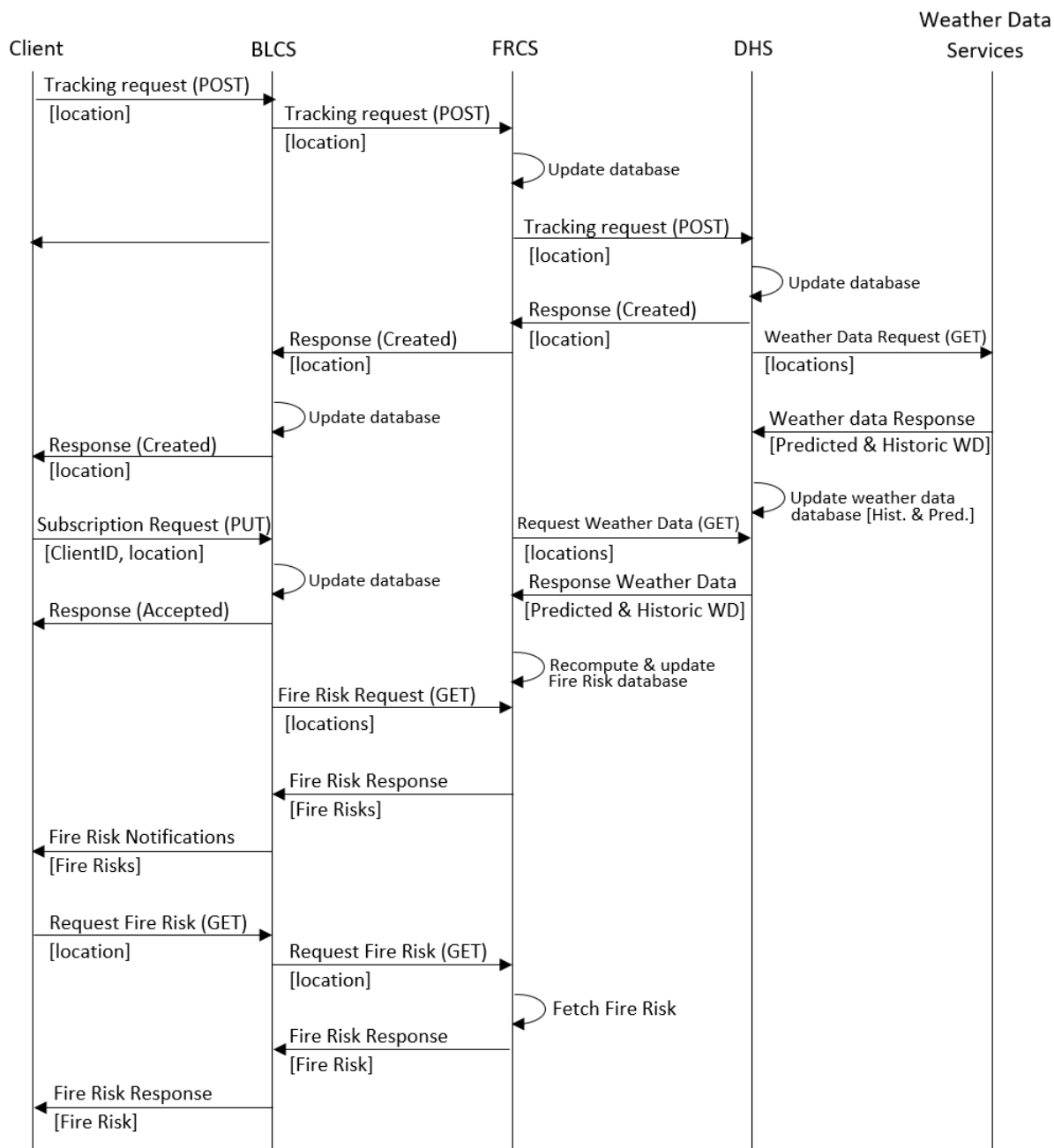


Figure 3: A sequence diagram for selected requests and processes, herein tracking, subscription and risk requests, as well as the scheduled update of databases and associated notification.

than the content, *e.g.*, the verification only requires to distinguish between the absence of a notification of fire risk or its sending, but not on the actual levels of risk. Hence, risk levels are

abstracted and emphasis is given to properly capture the purpose of the interactions between the constituent services. A consequence of this is that we also abstract from the actual values related to weather data and only consider the absence or presence of weather data for locations.

3. Business Logic Controller Service

The BLCS which is modelled by the submodule of the *Business Logic Controller Service* substitution transition (see Fig. 2) consists of two substitution transitions and associated sub-modules, as well as the subscription database. The two modules represent (1) the processing of client requests and (2) the scheduled update and associated notifications. Figure 4 shows the module responsible for the processing of client requests, the *BLCS Handle Request*. It is connected to the FRNS module in Figure 2 through the places with similar naming. It contains three transitions representing the different incoming client requests, that is, tracking, subscription and single fire risk request. A request either results in an immediate response in the case of a subscription request, or the requesting of further information from the FRCS in the case of tracking or single fire risk request, as can be seen from Figure 4.

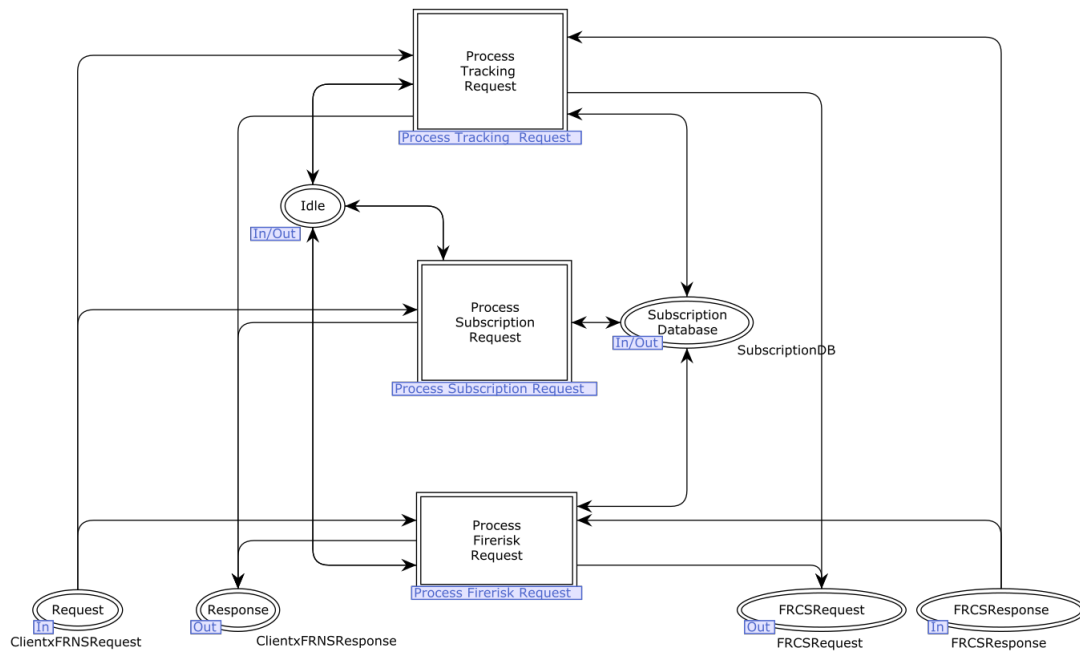


Figure 4: Presenting the sub-module responsible for handling requests within the BLCS, the *BLCS Handle Request*. Represented through respective substitution transitions, from the top, the handling of tracking, subscription and single fire risk requests.

Considering a request from the *Client Applications*, a token is produced onto the place *Request*, connecting this application with the BLCS. The colour set associated with this place is *ClientxFRNSRequest*, which is a product colour set combining the index colour set *Client*

```

1 colset Method      = with GET | PUT      | POST      | DELETE;
2 colset StatusCode = with OK | CREATED | ACCEPTED | NOTFOUND;
3 colset Component  = union BLCSTracking + BLCSUpdate + BLCSRequest;
4
5 val Cn = 2;
6 val Ln = 5;
7 colset Client      = index Client with 1..Cn;
8 colset Location    = index Loc   with 1..Ln;
9 colset Locations   = list Location;
10 colset ClientxLocation = product Client * Location;
11
12 colset FireRisk    = with Risk | NA;
13 colset LocxFireRisk = product Location * FireRisk;
14 colset FireRisks  = list LocxFireRisk;
15 colset FRCSResource = union FRCSFirerisks + FRCSLocation : Location;
16 colset FRNSResource = union FRNSLocation : Location + FRNSFirerisk : Location +
17 FRNSLocations : ClientxLocation + FRNSFirerisks : FireRisks;
18
19 colset FRNSRequest = record method : Method * resource : FRNSResource;
20 colset ClientxFRNSRequest = product Client * FRNSRequest;
21
22 colset FRNSResponse = record response : StatusCode * body : FRNSResource;
23 colset ClientxFRNSResponse = product Client * FRNSResponse;
24
25 colset FRCSReq = record method : Method * resource : FRCSResource;
26 colset FRCSRequest = product Component * FRCSReq;
27
28 colset FRCSResp = record response : StatusCode * body : FireRisks;
29 colset FRCSResponse = product Component * FRCSResp;

```

Figure 5: Colour sets definitions used to model the state of the BLCS.

with the record colour set *FRNSRequest*, as can be seen from Figure 5. Within the model, a request includes the specification of a HTTP method as well as the resources needed to handle the request. The resources are modelled through a union colour set, defining the REST API endpoints (constructors) and the associated data (arguments). The resources are service specific, defined for all interactions associated with the service. Hence, requests and responses use the same resource within respective services. With respect to the place *Request*, the resource is modelled by *FRNSResource*, and depending on the type of request and associated endpoint, the carried data is either a location or a combination of a location and a client identifier.

The colour set *ClientxFRNSResponse* associated with place *Response* is similar to *ClientxFRNSRequest*, but instead makes use of the colour set *FRNSResponse*. Within the model, a response is defined as a record set and includes a *response* and a *body*, associated with the colour sets *StatusCode* and the aforementioned *FRNSResource* respectively. The *response* is used to indicate the status post processing the request, while the *body* embeds response data, if any. At the place *Response*, depending on the request, the resource is either a client and a location or a location and a fire risk. For any request not specifically asking for a fire risk, the response (*StatusCode*) represents the data needed by the client.

If the BLCS needs to request the FRCS, a token is produced at the place *FRCSRequest*, while in response, a token will be received at *FRCSResponse*. The colour sets associated with these places have similar names as the places themselves, hence *FRCSRequest* and *FRCSResponse*. These colour sets follow the same general structure as previously described for request and response colour sets. However, distinguishing these union sets from the previously described colour sets, is the inclusion of a *Component*. The *Component* identifies the type of request and from where it originates. This can be used within the model to distinguish the different ongoing interactions, as well as being useful when performing single-step and interactive simulations. Then, the defined colour set becomes a union of the *Component* and the record set *FRCSReq* or *FRCSResp*, as can be seen in Figure 5. The colour sets *FRCSReq* and *FRCSResp* follow the previously presented design of requests and responses. The *FRCSReq* consists of a HTTP method and the associated resource *FRCSResource*, while *FRCSResp* contains a response described by the *StatusCode* and an associated body containing the explicitly stated *FireRisks* colour set.

To consider how a specific request is handled, Figure 6 presents the sub-module of *Process Tracking Request* from Figure 4. At the top left, the familiar place *Request* can be seen, as well as the other recently considered places at the remaining corners. Any client request, regardless of its type, will occur at the top left. However, only the POST requests associated with resource (endpoint) *FRNSLocation* will be considered within this module, hence the arc inscriptions. POST requests are associated with clients requesting tracking of new locations. A requested location tracking may either contain an unconsidered location or an existing location. Hence, guards at the transitions connecting to the place *Request* consider whether the requested location already exists. If the function *hasLoc* (has location) evaluates to true, it means the requested location already exists within the subscription database and only the transition *Request Location Exist* becomes enabled. If this is the case, a token is produced at the place *Response*, containing an ACCEPTED response and the location in question. Similarly, the transition *Process New Location* only becomes enabled if *hasLoc* is not true and there is an available token at place *Idle*.

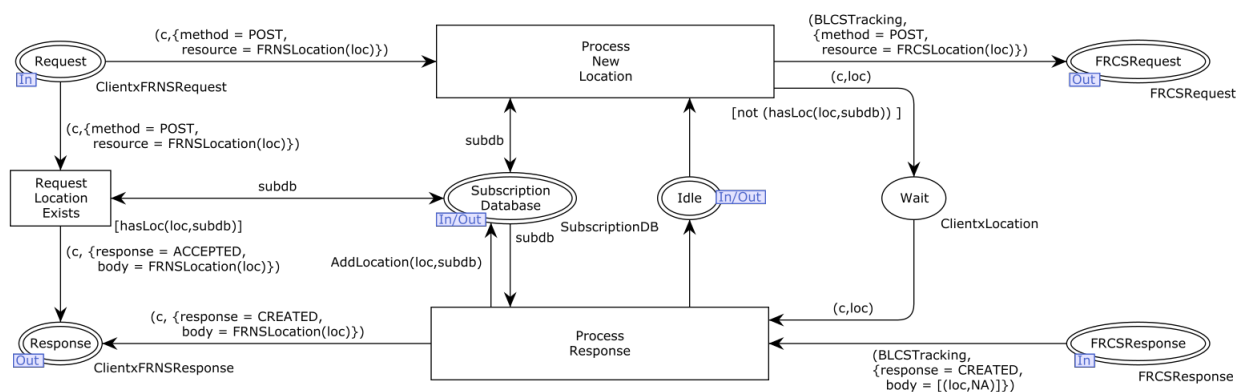


Figure 6: The *Process POST Request* submodule. Presenting the BLCS handling of a tracking (POST) request.

The Idle tokens are introduced to ensure the full consideration of a request or process, before engaging the next. This is in accordance with the prototyped system as well as contributing to a

reduced state space. When fired, the transition produces a token at *FRCSRequest* in accordance with its associated colour set. As can be seen from the arc inscription, the token consists of the component *BLCSTracking*, as well as the specified POST method and *FRCSLocation* resource.

When a tracking response is received at place *FRCSResponse* it contains the *response* CREATED, a body with the specific location and a *FireRisk* corresponding to NA (not available). Together with the wait-token produced upon the firing of *Process New Location*, the *FRCSResponse* token enables the transition *Process Response*, which in turn may update the subscription database through the function *AddLocation*, as well as producing a response to the place *Response*. Note that it is evident from the *component* that the response originates from a tracking request, hence the NA within the *body* of the response.

4. FireRisk Computation Service

The FRCS is responsible for the computation of fire risks and the handling of the *FireRisk DataBase*. Fire risks are computed by use of weather data received from the DHS and computed values are stored within the risk database, for later retrieval. The FRCS either responds to requests from the BLCS or performs scheduled recomputations of the fire risks within the database. Figure 7 presents the FRCS submodule, where the two substitution transitions *FRCSProcessRequest* and *Recompute FireRisks* represent the processing of requests and scheduled recomputations, respectively. The place *FireRisk DataBase* is associated with the colour set *FireRiskDB*, which is a list of entries consisting of the combination of *location* and *FireRisk*. The places *DHSRequest* and *DHSResponse* are constructed in accordance with the general design of requests and responses. The associated resource is modelled by *DHSResource*, which based on the endpoint (constructor), is either a location, a list of locations or a list of combinations of location and weather data, as can be seen from Figure 8.

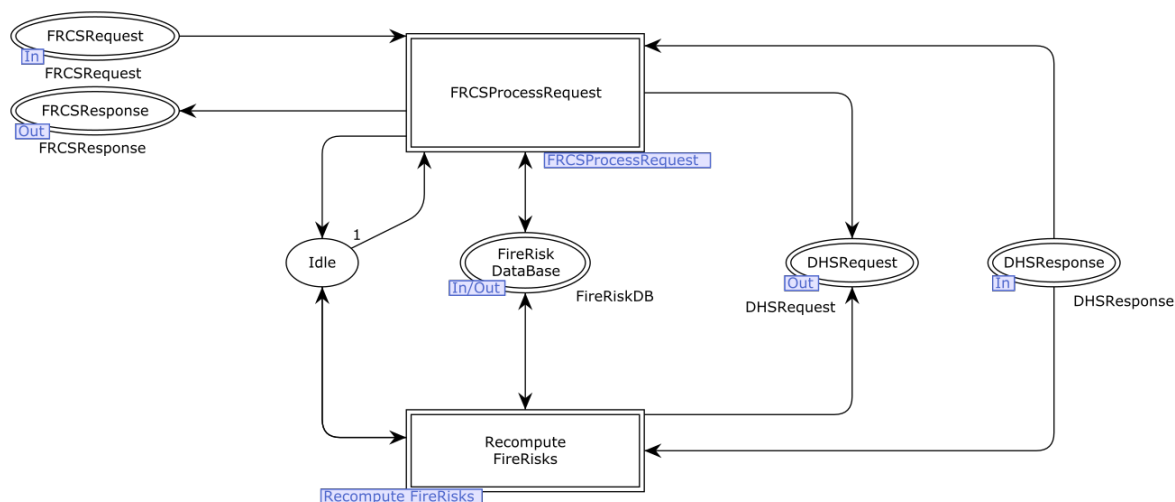


Figure 7: The *FireRisk Computation Service* submodule and associated substitution transitions and database.

Expanding the substitution transition *FRCSProcessRequest* gives the sub-module presented in Figure 9. The module consists of two substitution transitions, representing the processing of tracking requests (*Process Tracking Request*) and single fire risk requests (*Process GET Request*). These are the only two types of requests processed by the FRCS, as the subscription requests were limited to within the BLCS. If the recently considered tracking request from Section 3 is considered, it would appear at the place *FRCSRequest* to be processed within the substitution transition *Process Tracking Request*, which is similar to what was previously presented related to Figure 6.

```

1 colset WeatherData = with FORECAST | MEASUREMENT | NOTPRESENT;
2 colset LocxWDxWD = product Location * WeatherData * WeatherData;
3 colset LocsxWDS = list LocxWDxWD;
4 var wd : LocsxWDS;
5 colset DHSResource = union DHSLocation : Location + DHSLocations : Locations +
6 DHSWD : LocsxWDS;
7 colset DHSRequest = record method : Method * resource : DHSResource;
8 colset DHSResponse = record status : StatusCode * body : DHSResource;

```

Figure 8: Colour set definitions used to model the state of the FRCS.

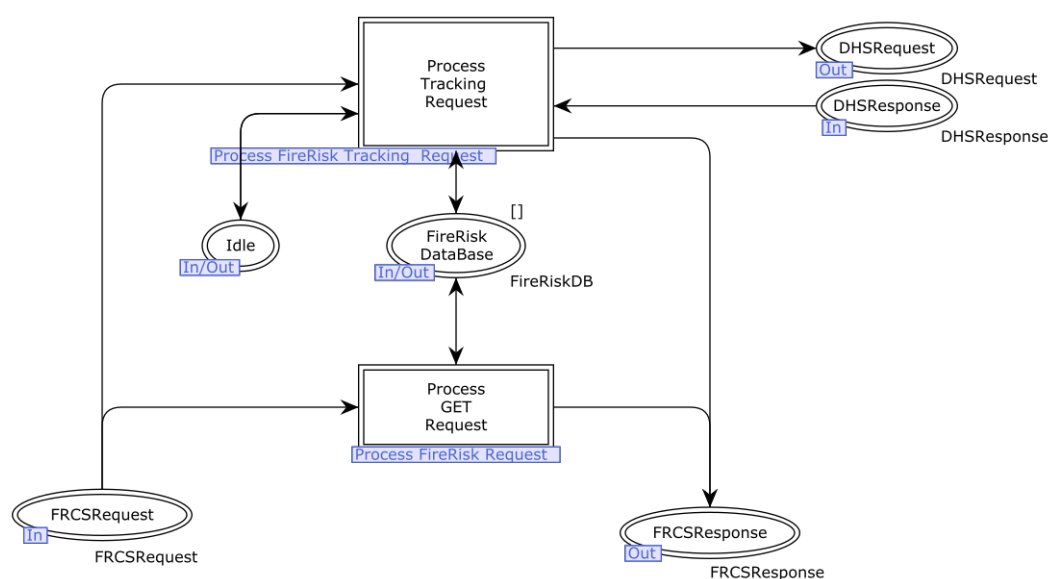


Figure 9: The sub-module of *FRCS Process Request*. It can be seen that the FRCS handles tracking (POST) and single fire risk (GET) requests only.

The handling of the scheduled update of the *FireRisk DataBase* is presented in Figure 10 and is the sub-module of the transition *Recompute FireRisks* from Figure 7. When firing, transition *Send Request* requests weather data from the DHS, for all locations kept within the fire risk database. The response received is a *status* OK and a list of locations and associated risks. The

latter resource is represented by the *DHSWD* endpoint (constructor) and the variable *wd*, as can be seen from the arc inscription and declarations in Figure 8. The received response results in the update of the *FireRisk DataBase* through the *updateFRDB* function.

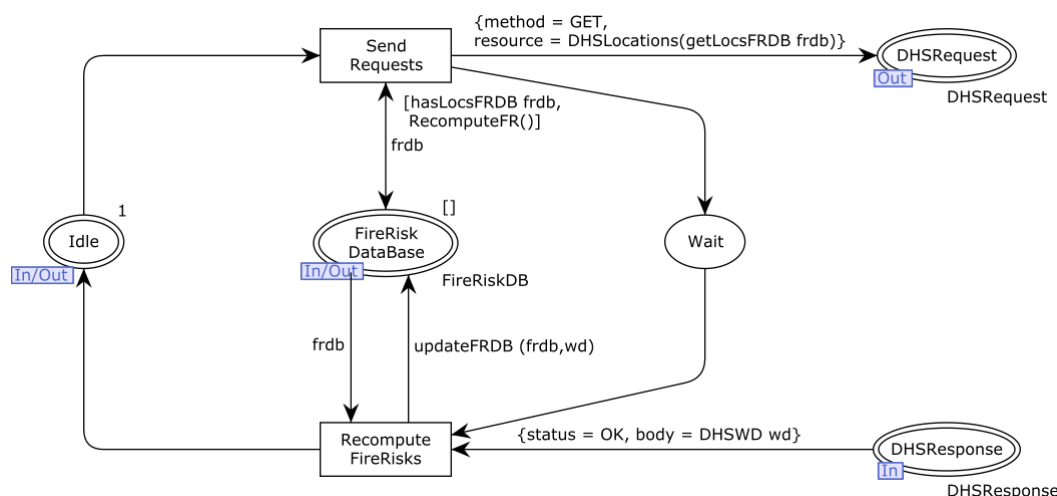


Figure 10: The sub-module *Recompute FireRisks* responsible for the scheduled update of the *FireRisk DataBase*.

5. Data Harvesting Service

The DHS is responsible for retrieving weather data from external sources and handling the *Weather Data DataBase*. The DHS either responds to requests from the FRCS or performs scheduled updates of the weather data in the database by requesting external services. The data fetched by the DHS is both historic and predicted weather data, requested from the FROST and MET APIs of the Norwegian Meteorological Institute, respectively. Figure 11 presents the DHS sub-module for processing requests. The three substitution transitions are *Process POST Request*, *Process GET Request* and *Process Delete Request*. The POST request is related to the initiation of location tracking and involves updating the local database, as previously described. The GET request is a request for service specific data, in this case weather data needed by the FRCS to perform fire risk computations. Yet unaddressed, is the delete request. Any subscription or tracking of a location can be terminated, which involves deleting clients or locations from the databases. Within the DHS, the delete request is only related to the termination of a tracking, hence the deletion of a location within the weather database.

The DHS is responsible for harvesting weather data from the external services. Figure 12 presents the modelling of the weather data harvesting, through a single harvesting module. The requesting of weather data from the two external services of FROST and MET, is modelled through a single combined weather data request. In turn, this means that the weather data response is a combined response, containing both historical and forecast weather data. The

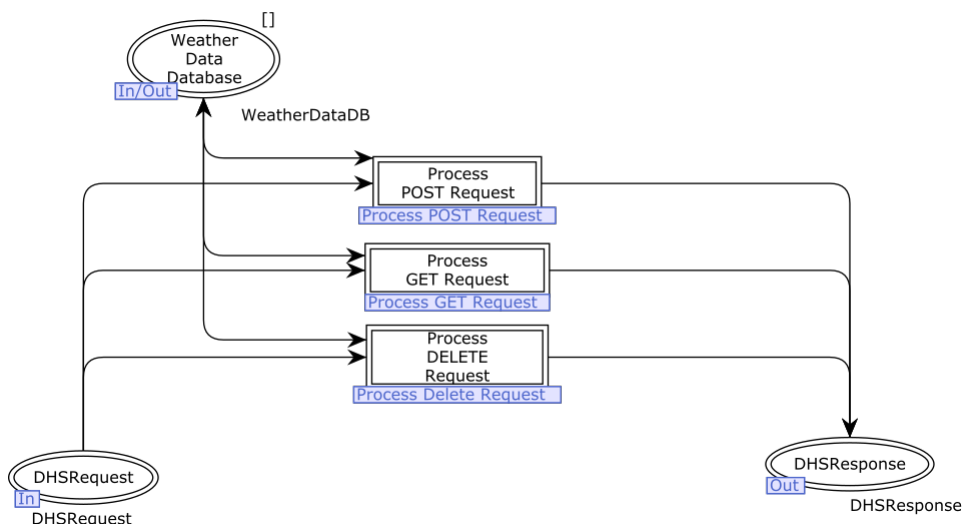


Figure 11: The sub-module responsible for handling requests within the DHS. The DHS either POST or DELETE a location from its database, or fetches weather data from the *Weather Data DataBase* (GET).

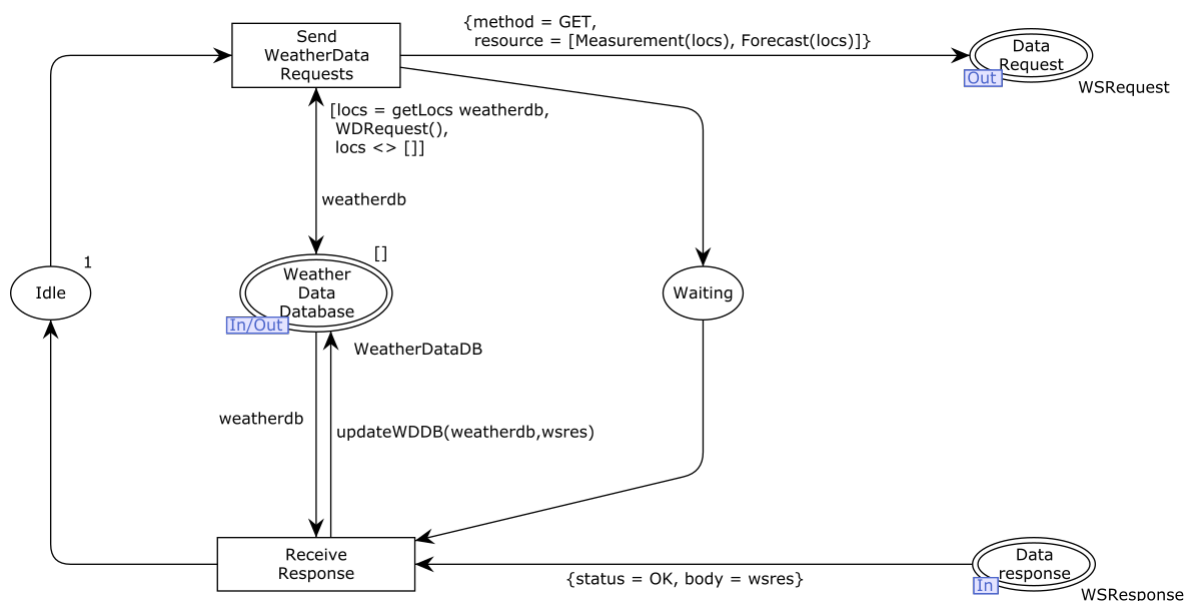


Figure 12: The sub-module *Harvest Weather Data*, within the DHS, responsible for the harvesting of weather data from external services. The modelling combines the requesting from two external services into a single combined request.

tokens produced, associated with the communication with the external services, appear at places *Data Request* and *Data Response*. The colour sets used are *WSRequest* and *WSResponse*

```

1 colset WSResource = union Forecast : Locations + Measurement : Locations;
2 colset WSRequest = record method : Method * resource : WSResource;
3 colset WSResponse = record status : StatusCode * body : WSResource;
4
5 colset WSResource = union Forecast : Locations + Measurement : Locations;
6 colset WSResources = list WSResource;
7 colset WSRequest = record method : Method * resource : WSResources;
8 colset WSResponse = record status : StatusCode * body : WSResources;

```

Figure 13: Colour set definitions used to model the state of the DHS.

and follow the aforementioned request and response structure. The request contains a HTTP method and a resource, while the response contains a status and a body. The resource is *WSResource* and represent both forecast and historic weather data for one or more locations, as can be seen from Figure 13. At last, a weather data response results in the update of weather data within the database, which in turn can be requested by the FRCS.

6. Model Validation and Verification

To validate the constructed CPN model, we initially performed single-step execution combined with interactive and automatic simulation. The aim of using simulation was to confirm that the basic behaviour of the CPN model was as intended, including the interaction between micro-services. Testing the CPN model using simulation identified a number of smaller modelling errors which could easily be corrected.

To perform a more exhaustive verification of the CPN model, we report in this section on how we have used the state space exploration facilities of CPN Tools in conjunction with the state- and action-oriented ASK-CTL library [24, 25] to verify key behavioral properties of the fire risk notification system using temporal logic. We adopted an incremental verification approach similar to the one developed in [26], where we gradually verified the services of the fire risk notification system. In particular, we started with the verification of the location tracking service and then incrementally enabled subscription, fire risk computation, and data harvesting until we finally considered the verification of some system-wide behavioural properties. In the following paragraphs we present the behavioural properties considered for the different services. We assume that the reader is familiar with the basic CTL temporal operators **AG** (always/globally) and **EF** (reachable/exists future). In the specification of the CTL properties, we use L to denote the set of locations and C to denote the set of clients.

Tracking properties. Clients send requests to start and stop the tracking of locations which in turn determines the locations for which weather data is being harvested and for which fire risks are being computed. For tracking we consider the following properties.

T-P1 $\forall c \in C, l \in L : \mathbf{AG EF} (c \text{ requests tracking of } l)$. This property states that it is always possible for any client to initiate the tracking of any location. The proposition $c \text{ requests}$

tracking of l can be implemented by considering the binding of the transition in the *Client* module corresponding to the event of requesting a location to be tracked.

- T-P2** $\forall c \in C, l \in L : \mathbf{AG} ((c \text{ requests tracking of } l) \Rightarrow \mathbf{EF} (l \text{ is being tracked}))$. This property states that if a client requests tracking of a location, then there exists a future state in which this location is being tracked. Checking the location can be implemented by considering the tokens present on the places representing the databases of the three micro-services.
- T-P3** $\forall c \in C, l \in L : \mathbf{AG} \mathbf{EF} (c \text{ stops tracking of } l)$. This is the dual property of T-P1 for stopping the tracking of locations.
- T-P4** $\forall c \in C, l \in L : \mathbf{AG} (c \text{ stops tracking of } l) \Rightarrow \mathbf{EF} (l \text{ is not being tracked})$. This is the dual property of T-P2 for stopping the tracking of locations.

Since some components in the CPN model are not synchronised, we consider the **EF** operator in the properties rather than the stronger **AF** (always eventually) operator. Extending our work to verify the stronger always eventually properties would require the application of fairness assumptions. This could be achieved by *e.g.* using linear temporal logic (LTL) and encoding the exclusion of non-fair executions in the formulae being verified. However, there is currently no support for LTL modelling in CPN Tools.

Subscription properties. Clients subscribe to a location in order to receive fire risk notifications for that location. For subscription we consider the following properties.

- S-P1** $\forall c \in C, l \in L : \mathbf{AG} \mathbf{EF} (c \text{ requests subscription to } l)$. This property states that it is always possible for any client to request subscription to any location. The proposition *c requests subscription to l* can be implemented by considering the binding of the transition in the *Client* module corresponding to the event of requesting a subscription.
- S-P2** $\forall c \in C, l \in L : \mathbf{AG} ((c \text{ requests subscription to } l) \Rightarrow \mathbf{EF} (c \text{ is subscribed to } l))$. This property states that if a client requests subscription to a location, then there exists a future state in which the client is subscribed to that location. Checking the client subscription to the location can be implemented by considering the tokens present on the *Subscription Database* place (see Figure 2).
- S-P3** $\forall c \in C, locs \subseteq L : \mathbf{AG} \mathbf{EF} (c \text{ is subscribed to all locations in } locs)$. This property states that it is always possible for a client to be subscribed to any subset of locations.
- S-P4** $\forall c \in C : \mathbf{AG} (c \text{ is subscribed to a subset of } L)$. This property ensures that a client cannot have multiple subscriptions to the same location.
- S-P5** $\forall c \in C, l \in L : \mathbf{AG} \mathbf{EF} (c \text{ unsubscribes to } l)$. This is the dual property of S-P1 but for unsubscribing to a location.
- S-P6** $\forall c \in C, l \in L : \mathbf{AG} ((c \text{ unsubscribes to } l) \Rightarrow \mathbf{EF} (c \text{ is not subscribed to } l))$. This is the dual property of S-P2 but for unsubscribing to a location.

Fire risk properties. Fire risk is to be computed by the fire risk computation service for a given location when this location is being tracked. For the fire risk computation service, we consider the following properties.

F-P1 $\forall l \in L : \mathbf{AG EF}$ (firerisk is computed for l). This property states that for any location it is always possible to compute the fire risk. The fire risk being computed for a given location can be checked from the *Firerisk Database* place.

F-P2 $\forall l \in l : \mathbf{AG EF}$ (firerisk recompute). This property states that it is always possible to recompute the fire risk for any given location, and hence that fire risk can be periodically recomputed. Re-computation of fire risks corresponds to the occurrence of the *Recompute FireRisks* transition in Figure 10.

F-P3 $\forall c \in C, l \in L : \mathbf{AG}$ (c requests firerisk for $l \Rightarrow \mathbf{EF}$ (c receives firerisk response for l)). This property states that if a client c requests a fire risk for a given location l , then it is possible for c to obtain the response.

Data harvesting properties. The data harvesting service is to retrieve weather data periodically for the currently tracked locations in order for the fire risk computation service to be able to compute fire risks. For data harvesting we consider the following properties.

W-P1 $\forall l \in L : \mathbf{AG EF}$ (weather data is stored for l). This property states that for any location it is always possible to store weather data. That the weather data has been stored for a given location can be checked from the *Weatherdata Database* place.

W-P2 $\forall l \in L : \mathbf{AG EF}$ (weather data requested for l). This property states that it is always possible to harvest data for a given location. The harvesting of data for a location corresponds to an occurrence of the *SendWeatherDataRequests* transition in Figure 12.

Inter-service properties. In addition to the properties related to specific services above, we also consider the following properties which rely on the collaboration between all micro-services in the system.

A-P1 \mathbf{AG} (no pending client requests \Rightarrow consistent data bases). This property states that if no client is currently executing requests against the service, then the databases of the micro-services are consistent in terms of storing information for the same set of locations. That the databases are consistent can be checked by considering the markings of the three places representing information stored in the databases (see Figure 2).

A-P2 $\forall c \in C, l \in L : \mathbf{AG}$ (c is subscribed to l) $\Rightarrow \mathbf{EF}$ (c receives fire risk notification for l). This property states that if a client is subscribed to a location, then it is possible for the client to receive fire risks for that location. That a client is subscribed to a location can be retrieved from the place representing the subscription database while the reception of a notification is represented by the occurrence of the corresponding transition in the *Client* module.

Experimental results. We verified the properties presented above for selected configurations of the CPN model in terms of locations, clients, tracked locations, and subscriptions.

Table 1 summarises the statistics for the state space exploration and verification of properties. The *States* and *Arcs* columns give the number of states and edges, respectively, in the state space. The *G-Time* column provides the time (in seconds) used to generate the state space for the given configuration while the *V-Time* column lists the time (in seconds) used for verification of properties. The verification was undertaken on i5-PC 2.4 GHz PC with a 16Gb memory.

We use $Cx - Ly$ to denote a configuration with x clients and y locations. A $*$ indicates that we have fixed the tracking and subscription such that all locations are tracked and all clients subscribe to all locations. We use *Request* to specify configurations where we only consider single requests from clients and *Notify* to specify configurations where we consider only the system initiated notifications (and not individual client requests). It can be observed that for configurations with only notification, the size of the state space does not grow when increasing the number of locations. This is due to the subscriptions being fixed and the fact that a client is notified about all subscribed locations in one single message.

Table 1

State space and verification statistics for configurations considered in the verification.

Configuration	States	Arcs	G-Time	V-Time
C1-L1	3,435	11,901	< 1 s	< 1 s
C1-L2	215,181	739,797	2,093 s	555 s
C2-L1	274,581	1,238,395	9,100 s	1,404 s
C2-L2-*	14,556	62,535	27.7 s	26.6 s
C2-L3-*-Request	5,151	21,138	4 s	10.5 s
C2-L3-*-Notify	216	687	< 1 s	< 1 s
C3-L2-*-Request	18,654	91,596	49.5 s	67.0 s
C3-L2-*-Notify	372	1,311	< 1 s	< 1 s
C3-L3-*-Request	54,894	276,378	480.9 s	339.8 s
C3-L3-*-Notify	372	1,311	< 1 s	< 1 s

The verification of the model revealed an error related to the notification of clients with respect to fire risks which was not identified during simulation. This demonstrates the added value of undertaken state space exploration of the CPN model, in order to perform more exhaustive verification of properties.

7. Conclusion

In this paper we have presented a formal specification of the micro-service based architecture for the fire risk notification system being developed in the DYNAMIC research project. Our

modelling patterns represent a general approach to modelling REST APIs using CPNs. In addition, we have formally validated the system services and the interaction between the micro-services using state space exploration and model checking.

The design of the fire risk notification system has been following an implementation-first approach where two prototypes were implemented and tested. The purpose has been to first validate the fire risk model itself, as to better understand the functional requirements and software technology capabilities in the technical solution space. The final step has then been to specify the software architecture and micro-services in an implementation-independent manner using a CPN model. The CPN model presented in this paper will then serve as basis for implementing a final version of the fire risk notification system.

In the present work we have considered only a limited number of system configurations. Future work includes verification of a more complete set of system configurations using the incremental methodology presented in [26]. As part of this, we may also investigate the use of fairness assumptions in the verification to be able to verify eventual-type properties and hence strengthen the properties being verified from the system. Furthermore, the constructed CPN model may potentially be used to generate test-cases for the implementation using the model-based approach presented in [27].

Acknowledgments

This study was partly funded by the Research Council of Norway, grant no 298993, Reducing fire disaster risk through *dynamic risk assessment and management* (DYNAMIC). The study was also supported by Haugaland Kraft Nett, Norwegian Directorate for Cultural Heritage and Stavanger municipality. Part of this work was achieved while Lars M. Kristensen was visiting Université Sorbonne Paris Nord as invited professor.

References

- [1] T. Log, Cold climate fire risk; a case study of the Lærdalsøyri fire, *Fire Technology* 52 (2014) 1815–1843.
- [2] DSB, Brannene i Lærdal, Flatanger og på Frøya Vinteren 2014 (The fires at Lærdal, Flatanger and Frøya, winter 2014), Technical Report, Norwegian Directorate for Civil Protection, 2014. In Norwegian.
- [3] S. Rohrer-Mirtschink, N. Forster, P. Giovanoli, M. Guggenheim, Major burn injuries associated with christmas celebrations: a 41-year experience from switzerland, *Annals of burns and fire disasters* 28 (2015) 71–75.
- [4] A. R. Pirsko, W. L. Fons, Frequency of Urban Building Fires as Related to Daily Weather Conditions, Technical Report 866, US Dep. of Agriculture, 1956.
- [5] T. Log, Indoor relative humidity as a fire risk indicator, *Building and Environment* 111 (2017) 238–248. doi:<https://doi.org/10.1016/j.buildenv.2016.11.002>.
- [6] T. Log, Modeling indoor relative humidity and wood moisture content as a proxy for wooden home fire risk, *Sensors* 19 (2019). doi:10.3390/s19225050.

- [7] A. Kraaijeveld, A. Gunnarshaug, B. Schei, T. Log, Burning rate and time to flashover in wooden 1/4 scale compartments as a function of fuel moisture content, *Proceedings of the 14th Int. Fire Science and Eng. Conf, Interflam, Windsor, UK*, pp. 553-558, 4-6 July, 2016.
- [8] M. Metallinou, T. Log, Cold Climate Structural Fire Danger Rating System?, *Challenges* 9(1) (2018) 12.
- [9] Frost api, 2022. URL: <https://frost.met.no/index.html>, (accessed: 27.03.2022).
- [10] Met norway weather api, 2022. URL: <https://api.met.no/>, (accessed: 27.03.2022).
- [11] Reducing fire disaster risk through dynamic risk assessment and management (dynamic), 2022. URL: <https://www.hvl.no/en/project/2495578/>, (accessed: 27.03.2022).
- [12] R. D. Strand, S. Stokkenes, L. M. Kristensen, T. Log, Fire risk prediction using cloud-based weather data services, *Journal of Ubiquitous Systems & Pervasive Networks* 16 (2021). doi:10.5383/JUSPN.16.01.005.
- [13] Heroku, 2022. URL: <https://devcenter.heroku.com/articles/heroku-cli>, (accessed: 28.03.2022).
- [14] Spring boot framework, 2022. URL: <https://spring.io/projects/spring-boot>, (accessed: 28.03.2022).
- [15] Xamarin framework, 2022. URL: <https://dotnet.microsoft.com/en-us/apps/xamarin>, (accessed: 28.03.2022).
- [16] M. Camilli, C. Bellettini, L. Capra, M. Monga, A formal framework for specifying and verifying microservices based process flows, in: *Software Engineering and Formal Methods - SEFM 2017 Collocated Workshops: DataMod, FAACS, MSE, CoSim-CPS, and FOCLASA*, volume 10729 of *Lecture Notes in Computer Science*, Springer, 2017, pp. 187–202. URL: https://doi.org/10.1007/978-3-319-74781-1_14. doi:10.1007/978-3-319-74781-1_14.
- [17] M. Camilli, C. Bellettini, L. Capra, A high-level petri net-based formal model of distributed self-adaptive systems, in: *Proceedings of the 12th European Conference on Software Architecture: Companion Proceedings, ECSA, ACM*, 2018, pp. 40:1–40:7. URL: <https://doi.org/10.1145/3241403.3241445>. doi:10.1145/3241403.3241445.
- [18] J. H. Röwekamp, M. Buchholz, D. Moldt, Petri net sagas, in: *Proceedings of the International Workshop on Petri Nets and Software Engineering 2021 co-located with the 42nd International Conference on Application and Theory of Petri Nets and Concurrency (PETRI NETS, volume 2907 of CEUR Workshop Proceedings, CEUR-WS.org, 2021*, pp. 65–84. URL: <http://ceur-ws.org/Vol-2907/paper4.pdf>.
- [19] M. Sakai, K. Takahashi, S. Kondoh, Method of constructing petri net service model using distributed trace data of microservices, in: *22nd Asia-Pacific Network Operations and Management Symposium, APNOMS, IEEE*, 2021, pp. 214–217. URL: <https://doi.org/10.23919/APNOMS52696.2021.9562589>. doi:10.23919/APNOMS52696.2021.9562589.
- [20] L. Kallab, M. Mrissa, R. Chbeir, P. Bourreau, Using colored petri nets for verifying restful service composition, in: *On the Move to Meaningful Internet Systems. OTM 2017 Conferences - Confederated International Conferences: CoopIS, C&TC, and ODBASE*, volume 10573 of *Lecture Notes in Computer Science*, Springer, 2017, pp. 505–523. URL: https://doi.org/10.1007/978-3-319-69462-7_32. doi:10.1007/978-3-319-69462-7_32.
- [21] M. Narayanan, A. K. Cherukuri, Verification of cloud based information integration architecture using colored petri nets, *International journal of computer network and information security* 2 (2018) 1–11. doi:10.5815/ijcnis.

- [22] M. Ndiaye, J.-F. Pétin, J.-P. Georges, J. Camerini, Practical use of coloured petri nets for the design and performance assessment of distributed automation architectures, in: L. Cabac, L. M. Kristensen, H. Rölke (Eds.), *Proceedings of the International Workshop on Petri Nets and Software Engineering 2016*, 2016, pp. 113–131.
- [23] A. Evjenth, E. Halderaker, Development and evaluation of a software system for fire risk prediction, Master's thesis, Western Norway University of Applied Sciences, 2021.
- [24] A. Cheng, S. Christensen, K. H. Mortensen, Model checking coloured petri nets - exploiting strongly connected components, *DAIMI Report Series 26* (1997). URL: <https://tidsskrift.dk/daimipb/article/view/7048>. doi:10.7146/dpb.v26i519.7048.
- [25] S. Christensen, K. H. Mortensen, *Design/CPN ASK-CTL Manual*, Version 0.9, 1996.
- [26] A. Rodríguez, L. M. Kristensen, A. Rutle, Formal modelling and incremental verification of the mqtt iot protocol, *Trans. Petri Nets Other Model. Concurr.* 14 (2019) 126–145.
- [27] R. Wang, L. M. Kristensen, V. Stolz, MBT/CPN: A tool for model-based software testing of distributed systems protocols using coloured petri nets, in: *Verification and Evaluation of Computer and Communication Systems - 12th International Conference, VECoS 2018*, volume 11181 of *Lecture Notes in Computer Science*, Springer, 2018, pp. 97–113.