Service Assurance Process Re-Engineering Using Location-aware Infrastructure Intelligence

Hani Jamjoom
IBM Watson Research

Nikos Anerousis
IBM Watson Research

Raymond Jennings
IBM Watson Research

Debanjan Saha
IBM Watson Research

***Abstract*—The continuous introduction of converged services such as VoIP and Video-On-Demand has created many operational challenges for service providers. In this paper, we describe how to use location-aware technologies, not only to integrate disparate management applications, but also to transform the underlying process to use geographical views as the focal point of management operations. Based on an engagement with a large Cable provider, we have designed and implemented 3i—Integrated Infrastructure Intelligence—to address key issues in the service assurance process. 3i is highly componentized and provides an intuitive way for creating role-based views through dynamic scoping, event aggregation and status projection, and location-driven active probing. We analyze the current service assurance process and compare it with the improved process after introducing 3i. Overall, the re-engineered process offers base execution improvements in alarm collection, problem drill-down and reporting, as well as complexity improvements throughout. 3i is a fully implemented tool and has demonstrated capabilities beyond its original intended scope as a decision support tool in planning and marketing functions.**

I. Introduction

In the recent years, the Cable and Telecom industries have rolled out a plethora of new and converged services such as VoIP, Video-On-Demand, and On-line gaming. One of the greatest challenges that many of these providers face is how to integrate these new services (and their associated management processes) into their existing operations. This challenge arises not only from the need to introduce new components (or devices) into their infrastructure, but also from the need to modify the corresponding processes for managing these services, and do this continuously as newer services are introduced.

Because of the high cost of achieving proper integration, what results is the addition of new management applications to manage the new service offerings. Not surprisingly, this decreases the efficiency of the overall management process as operation engineers or technicians[1] are required to touch more applications, each providing a subset of the end-to-end infrastructure. It also complicates root-cause analysis as disparate information—spread across the different management applications—need to be correlated for all components that can be affected by various failure scenarios.

For many Cable providers, addressing the integration challenges at the process and IT infrastructure levels is of paramount importance in the areas of service fulfillment, assurance, and billing. In this paper, we describe how to use location-aware technologies, not only to integrate disparate systems, but also to transform the underlying process to make this location-aware technology the focal point of operation. While location-aware technologies and Geographic Information Systems (GIS)[2] have been used for many years to capture and monitor various types of infrastructures [18], in this paper, we describe the use of GIS-based methodology to address three key challenges. The first challenge is to provide customized role-based views from the large number of managed components. The second challenge is to minimize the number of (process) steps required to manage the large number of (potentially correlated) alarms. The third challenge is to use GIS-based system to minimize the delay of executing active probes during interactive diagnosis. We show that addressing these challenges improves the efficiency of the underlying process.

We focus on a subset of the service assurance process that relates to infrastructure monitoring. We will refer to it as *Alarm-driven Service Assurance (ADSA)*. Within the eTOM framework [13], this process covers aspects of three specific processes: *Survey and Analyze Resource Trouble*, *Support Resource Trouble Mgmt*, and *Track and Manage Resource Trouble*.

The ADSA process is conceptually straightforward. A major source of inefficiency is caused by the size of the monitored infrastructure and volume of alarms. These inefficiencies are most apparent in the time between problem occurrence and detection, which, even with automation and push-type mechanisms can be on the order of tens of minutes. There are four key causes for these inefficiencies. First, because of the large number of devices that are being monitored, management applications are typically configured to update their status every several minutes. Second, because monitored components are interdependent, many alarms are generated for the same failure. Detecting the actual failure requires sifting through large amounts of alarm information. Third, many perceived failures are self-correctable (e.g., power outages or hardware rests). This encourages technicians to wait for one or more monitoring cycles to eliminate self-correcting errors. Finally, because concurrent failures are common, lower-priority failures can be ignored for many minutes before a

[1] In this paper, we will refer to the role player responsible for monitoring as an operation engineer or technician

[2] In this paper, we will use the terms location-aware and GIS-based technologies interchangeably

1-4244-0799-0/07/$25.00 ©2007 IEEE

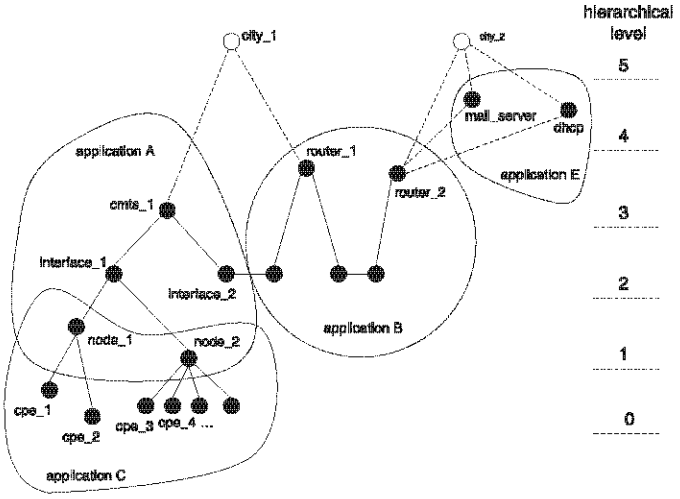

Fig. 1. Example topology, with filled circles representing real components and empty circles representing logical one. Similarly, a solid line represents a physical pairing and dashed line represents logical pairing.

technician is able to handle them. We are thus interested in minimizing the overhead (in term of delay and required human resources) that is due to the above.

To avoid confusion with commercial problem management solutions, we distinguish between alarm management and problem (or incident) management. In the former, we are referring to the relatively informal process of examining different pieces of an infrastructure and reacting to alarms generated by the monitoring tool. Problem management, on the other hand, involves a stricter process and typically uses systems that incorporate customer service-level agreements (SLA), e.g., ManageNow [8] or Remedy [2].

Our work has been motivated by a direct engagement with a large U.S. cable operator. The solution, called *3i—Integrated Infrastructure Intelligence*, is fully implemented and operational since early 2006. 3i offers an efficient way of managing the service assurance process of large distributed infrastructures.

This paper is organized as follows. Section II describes the service assurance process before the introduction of 3i. We then describe the architecture and implementation of 3i in Section III. In Section IV, we describe the resulting process and highlight efficiency improvements. The paper ends with related work in Section V and concludes in Section VI.

II. Current Process

In this section, we describe the current service assurance process. Our derived process is not specific to the Cable monitoring environment. Instead, it is derived from four key observations that we have encountered while interacting with multiple Cable and Telco operators; thus, we believe that they are representative of other infrastructure monitoring environments.

First, multiple management applications are used to monitor different pieces of the entire infrastructure. Considers the set of all components being monitored—which includes both active and passive devices—as vertices V in a connected graph $G_{topology}$, representing the monitored topology. The connected graph $G_{topology}$ also includes a set of edges $E_{physical}$ and $E_{logical}$ representing physical and logical pairings, respectively. Figure 1 shows an example topology, with solid lines representing physical pairings and dashed lines representing logical pairings. The need for logical pairings will be apparent shortly. In the mean time, we consider a management application, app, as monitoring a subset of the monitored component; each application, thus, forms a vertex-induced subgraph $G^{app}_{topology}$. In Figure 1, there are four subgraphs, each represented by the vertices and edges being monitored by the four applications.

Second, each subgraph $G^{app}_{topology}$ is hierarchically organized, reflecting some structuring of information. In Figure 1, we structure the monitored topology into six layers. However, each application covers a portion of these layers. This structuring can be derived from physical, logical, or spatial relationships between components. Keeping in mind that management applications must provide users with robust information navigation, we see the relevance of $E_{logical}$ above, which are used for two purposes: (1) create new relationships between components (e.g., group components within a specific city together) and (2) reconnect the monitored components which otherwise would form a disconnected subgraph if only physical pairings are used.

Third, different applications are often configured with different *name-spaces*. We use the term *name-space* to describe the way each application refers to the various monitored components. Applications A and C in Figure 1 may use different names to refer to *node_1*. In fact, it is quite common that different applications do not follow a well-agreed upon naming convention for monitored components, even when different applications monitor the same components. This complicates the diagnosis process by mandating (either implicitly or explicitly) a name-space translation step when navigating between one application to another. It is also a common source of misconfiguration issues.

Fourth, information from different applications relate to each other via a path in $G_{topology}$. For instance, *interface_1* in Application A relate to *cpe_2* in Application C by the path {*cpe_2*,*node_1*,*interface_1*}. We have observed that there is no formal way (e.g., shortest path) to capture this relationship, which, as we will describe later, is an important step in alarm diagnosis. Instead, it is relatively *ad hoc*, focusing primarily on matching name-spaces between different applications.

Based on the above four observations, we use Stochastic Petri-Nets (SPN) [16] to model the observed process for

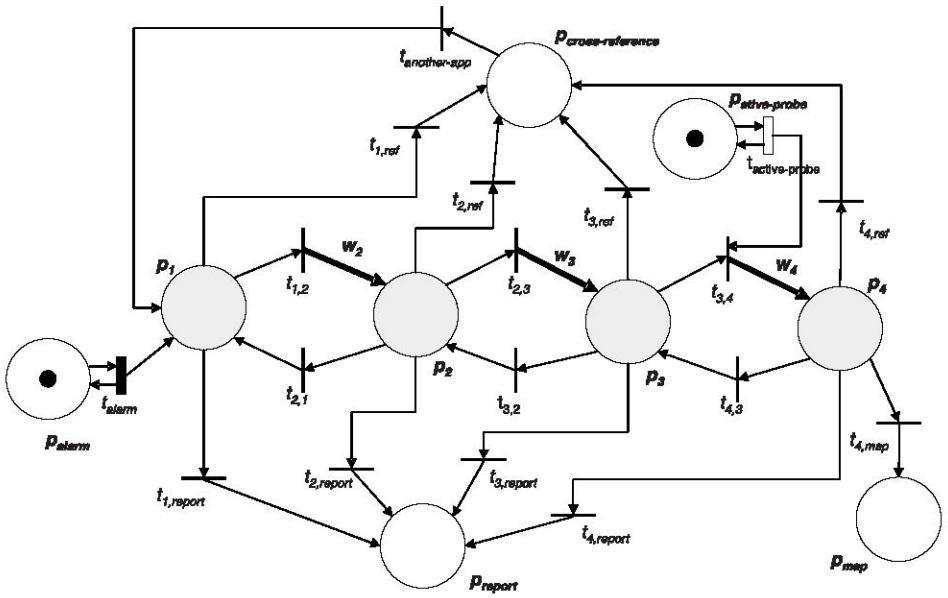

Fig. 2. Stochastic Petri-Net representation of the Alarm-driven Service Assurance process

monitoring the cable infrastructure.[3] Because the underlying process have a large degree of concurrency, SPN's provide an efficient way for capturing and analyzing the corresponding state machine. The basic process is depicted in Figure II. The places in the figure can be organized into four groups: (1) alarm generation (includes p_{alarm} and $p_{active-probe}$), (2) alarm drill-down (includes p_1, p_2, p_3, & p_4), (3) application-cross referencing (includes $p_{cross-reference}$), and (4) reporting (includes p_{report} & p_{map}). The remainder of this section looks at each group in more detail, highlighting the major source of inefficiencies.

A. Alarm Generation

Alarm generation includes both active and passive measurement. Because alarms are typically updated (or generated) after the completion of periodically-scheduled (passive) measurements of the infrastructure, these alarms arrive in batches, with periods ranging between 2 to 5 minutes. They are represented by new tokens in p_{alarm}. As noted earlier, because of the labor intensive requirement of tracking an alarm, it is customary for a technician to delay investigating the root-cause of the alarm for one or two measurement cycles to eliminate the possibility of transient problems. To that extent, we assume that alarms in p_{alarm} are persistent, requiring manual intervention to resolve their root-cause. As a corollary, the volume of alarms (number of tokens) is lower than if we also consider transient ones, which would only generate additional work for technicians. Finally, given the periodicity of alarm generation, we model t_{alarm} (the transition between alarm generation and the drill-down process) using a discrete value with parameter D.

While p_{alarm} reflects passive measurements, infrastructure monitoring also includes an active measurement component, focusing on end-devices like cable modems and set-top boxes. This is captured by $p_{active-probe}$ in Figure II. As we will describe shortly, active probes are initiated as part of the drill-down process. An active probe, while clearly provides more up-to-date information than its passive counterpart, incurs two penalties. First, results are not instantaneous, with delays reaching several minutes. Second, it requires substantial resources on the back-end servers. Unlike the transition t_{alarm}, $t_{active-probe}$ seems to be a function of the number of probed devices.

The above also highlights the information synchronization problem that arises from the fact that different applications can operate over different timescales. This is not just limited to information coming from passive measurements (i.e., the information is trailing the real status of a device), but also includes active measurements. Because of the large number of devices being monitored, even active probes can take long enough times that the information they provide is stale (especially, for transient errors).

B. Alarm Drill-Down

The drill-down steps mirror the *de facto* approach of structuring monitored devices into different layers (Figure 1). Although the number of layers can be arbitrary, we have observed that having three to five layers (per management application) is a standard practice (hence our use of four layers in describing the process). The first layer typically reflects

[3]For readers who are unfamiliar with Petri-Nets, they can concisely capture non-deterministic execution of state machines. They consist of places, transitions, and arcs. Each place can have one or more tokens (with the distribution of tokens across all places is referred as a marking). A transition can fire when there are enough tokens in the corresponding input. When the transition fires it consumes the token and produces one in the output place. A delay can be associated with each transition. Typically, a filled rectangle represents a deterministic delay, an empty rectangle represents a stochastic one, and a line represents no delay. Finally, each arch can have a weight. The weight reflects the number of tokens that a transition consumes or produces.

a geographical or physical grouping (e.g., region or device type). Thus, an alarm (or token) at level p_1—the starting point—reflects the highest level of alarm aggregation. Another common logical grouping is based on sensor type. A specific device, for instance, can be observed by multiple metrics such as power, signal-to-noise ratio, temperature, etc. It is not uncommon for multiple sensors to simultaneously indicate an alarm. Especially for open source tools, we have observed a lack of automation to correlate different sensor information, requiring further drill-down to relate the information.

Because of this hierarchical structuring of information, topologically higher-level components reflect aggregate status of child components. Thus, aggregation functions can affect the (observed) arrival of alarms. We have observed the use of two aggregation functions, one for active components of the monitored graph and the other for passive ones.

- In the case of a component being actively probed, a threshold-based aggregation function is commonly used, with typically two thresholds indicating a yellow (attention) or red (critical) alarm. For example, when monitoring the number of online modems on a specific interface, if we assume a yellow threshold of 90% and a red threshold of 80%, then at least 20% of modems need to go offline before a red alarm is generated.
- In the case of passive components, a boolean function is typically used. This is because a passive (parent) component (for example, the status of a city) mirrors the status of all child components. In this example, the status of the city is indicated as red if one or more of its children are red, with red having precedence over yellow alarms.

Looking at the SPN, the drill-down process is multiplicative: as one drills-down, additional tokens can be generated. These are indicated by the weight of the transition edges, which are functions of two variables: (1) the branching factor between a parent an its child components in the monitored subgraph of an application, and (2) the failure probability of a child component. Here, we assume that in absence of external events (e.g., power outage, storms, flooding, etc), components fail independently. This assumption holds true even when a component is passive and uses the aggregate status of its child components, because two parents at the same hierarchical level will typically form two disjoint subtrees. Not surprisingly, because external events can impact large areas, these failures are not as independent as one would hope. Nonetheless, assuming a Poisson failure process (a counting process $N(t)$ with stationary increments), we can describe w as follows:

$$w_l = E[N_l(t)] = \lambda_l t \tag{1}$$

where λ_l is the failure rate at a hierarchical level l. In many ways, we are oversimplifying things here as different devices can have different failure probabilities. Nonetheless, the equation above is intended to illustrate the time and hierarchical level dependencies that are part of the drill-down process.

For example, if we assume a single red alarm arriving in p_{alarm} indicating a critical problem in one or more of the monitored cities. Drilling down from p_1 to p_2 may generate additional tokens in p_2, indicating that some of the underlying components have failed. The technician would then have to individually drill down into each component, consuming one token from p_2 and generating additional tokens in p_3 to indicate that even more subcomponents have failed (e.g., an interface in Figure 1). At that point, an active probe on each failed interface is invoked, generating additional tokens in p_4.

While we cannot present exact figures, it is common for several interfaces to indicate some form of an alarm at any instance in time. Part of the reason is explained by Eq. 1, which when combined with actual failure probabilities yields consistent results.

C. Cross-Referencing

So far we have assumed that drilling down is confined to a single subgraph that is being monitored by a single application. In reality, an alarm is a symptom of a problem which is manifesting itself across multiple components that exist in different application subgraphs [7, 14]. That said, cross-referencing different applications (manually) to validate an assumption about the root cause of a problem is common practice. Cross-referencing, however, involves translating the context of one application to a different application. The translation of context would naturally involve a lookup step (depicted by $p_{cross-reference}$) in Figure II.

This step, informally, matches the name-space of one application to that of the application to be cross-referenced. For example, if one application is monitoring end-services (e.g., Video-On-Demand) and another application is monitoring the digital set-top box (STB) infrastructure, then a lookup on a given STB should return those VoD servers providing content to that STB. In many cases, this lookup process is learned through experience. In other cases, like for location information of a MAC address, more involved lookup might be required. In Figure II, we indicate that after the cross-reference lookup is performed, it transitions back into p_1 (however now it refers to the application to be cross-referenced). Because applications have few entry points, this transition points to the top-level of the new application (p_1), requiring some navigation to get to the needed information.

D. Reporting

The final set of steps involves report generation. They include p_{report} and p_{map}. They can also be considered the natural termination points of the process even if no explicit output is generated. We distinguish between two types of reports: (1) topological reports, which are based on the logical relationships between components (e.g., interface report for a specific sensor), and (2) physical or location-based reports which show the geographical map of specific elements. In our engagement, we have observed that most applications focus on topological reports and limited location-based reports (in most instances focusing on mapping individual devices).

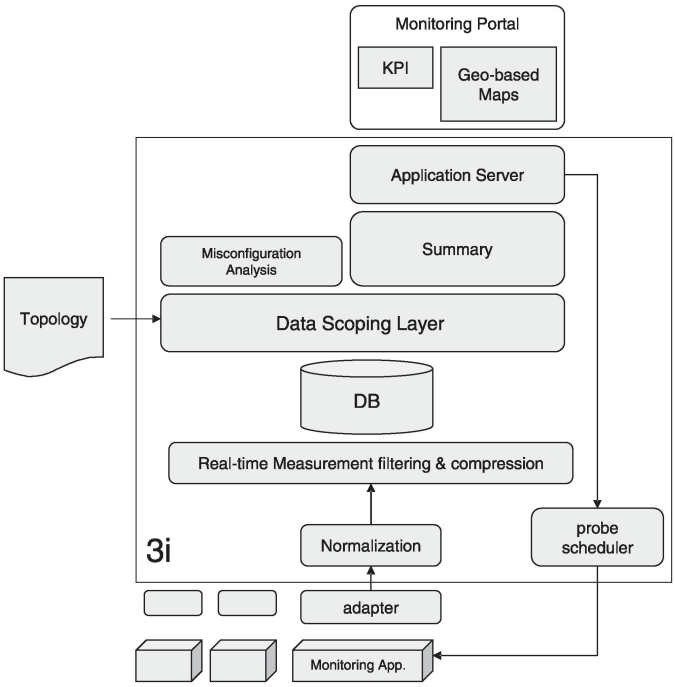

Fig. 3. Architecture of 3i.

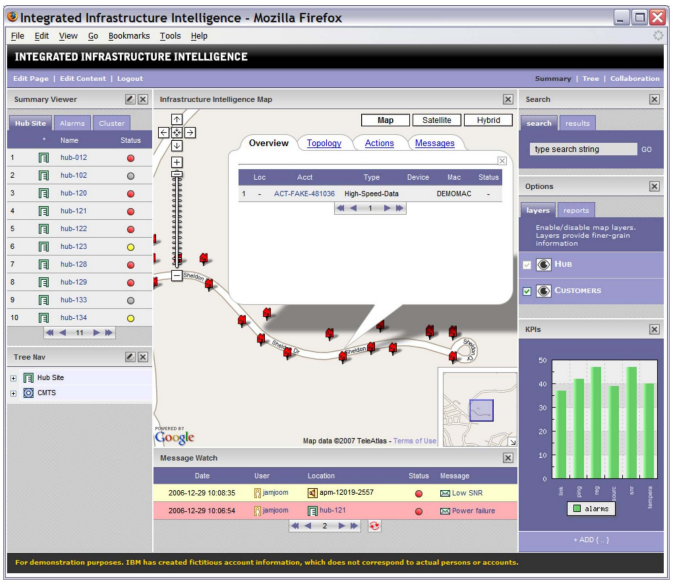

Fig. 4. Screen-shot of 3i

E. Reflecting on the Current Process

The described process captures the essential elements for alarm monitoring, with most inefficiencies originating from the cumbersome process of correlating alarms across different application and manual process of invoking active probes. There is another important element which we have not described in our process, namely, the role of information accuracy. As expected, information accuracy plays a critical role in the identification of root-causes. Unfortunately, information inaccuracies are common, with two types of inaccuracies of particular interest: (1) misconfiguration and (2) name-space errors. These inaccuracies are often caused by database corruption, disparate application configuration, infrastructure upgrades, etc. In both cases, we have observed the lack of automation in detecting and correcting such inaccuracies, relying on the technician's field knowledge to account for such inaccuracies. In Figure II, we have not explicitly captured the process of dealing with such inaccuracies. However, one can assume that they are part of p_1, p_2, p_3, p_4, and $p_{cross-reference}$.

III. Design & Implementation

To address the inefficiencies in the *current* process, we have fully implemented an application called *3i* that projects logical structuring of end-to-end infrastructure graph on top of a geo-coded map [11]. While contextualizing events in a Geographic Information System (GIS) is not new [18], this paper describes the use of GIS-based methodology to provide the following three features: (1) dynamic scoping based on user-configurable functional maps, (2) spatial event aggregation and projection, (3) minimizing overhead of active probing.

Before describing each feature, we briefly highlight the main components of 3i. Figure 3 shows the architecture of 3i, which follows similar software stack to other monitoring solutions like Micromuse [17] and SMARTS [5]. Figure 4 shows an anonymized screen-shot of 3i. At the lowest level is the event collection fabric. Here, events are assumed to come from other management applications. Events are normalized (i.e., converted into a standard format), compressed (by removing duplicate information) and persisted into a database (DB). The middle layer includes the core components that achieves the above features. They are described in the remainder of this section. The remaining components in Figure 3 are essentially responsible for packaging the information and sending it to the user front-end. In our implementation it is based on Google Maps API [6].

A. Dynamic Scoping

Combined with the topology information, the active DB has a complete view of the end-to-end graph G. This bring us to the first feature: *dynamic scoping*, which focuses on producing a consistent subgraph of G. Dynamic scoping addresses the requirement that different role-players might be interested in different types of information as well as different levels of aggregation.

A data scoping layer is introduced (Figure 3). It exposes a subgraph of G that can be scoped across three dimensions: (1) functionality, (2) time, and/or (3) space. Scoping by functionality allows a role player to define the type of components and sensors s/he is interested in. It operates at the functional map level. For instance, a technician can express interest in monitoring *REG* events from the Cable Modem Termination System (CMTS) interfaces (Figure 5). Scoping by time allows a role player to define a time-window of interest, which would effectively show events within the specified window. Finally,

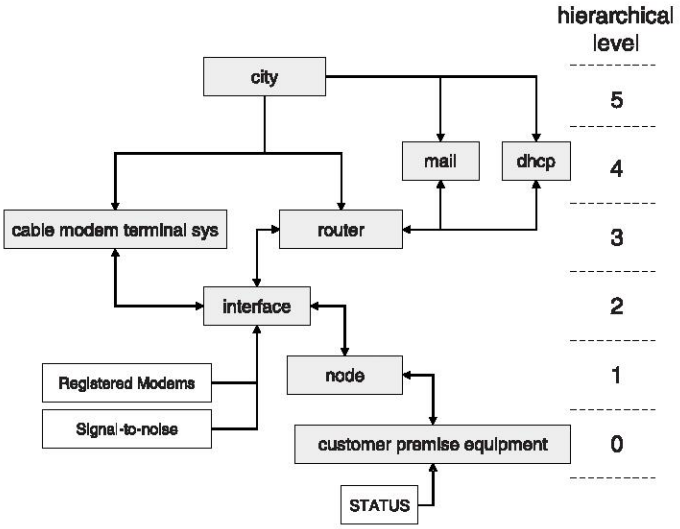

Fig. 5. Example of a functional map. Each row of *shaded* component types constitute a level in the hierarchical structuring o of information (total of five levels). A *non-shaded* box reflect different sensors. A component is considered *active* if there is a sensor connected to it, and *passive* otherwise.

scoping by space allows a role player to look at a components (and related events) within a specific geographical bound.

Implementing the above across the three dimensions is straightforward especially given capabilities of advanced databases like DB2 and Oracle (using Quadtrees indexing for fast lookups [22]). Where these databased fall short is in ensuring that the scoped graph is connected, especially since it is common for a scope to produce a disconnected subgraph of the end-to-end graph. Thus, the second role of the data scoping layer is to reconnect missing links between nodes (due to data scoping). It basically works by introducing logical connections when a child-parent chain is broken. It does not re-introduces missing components into the graph.

At this point, we can see that graph connectivity can be guaranteed if every node has at least one ancestor within the scope. Typically, only the spatial scoping dimension produces disconnected graphs. Because spatial scoping is used during map navigation, the resulting disconnected graph is still useful.

Figure 6 shows an example of how the data-scoping layer re-connects the scoped graph. The algorithm takes the full topology as its input and iterates over all of the vertices in the scoped graph as follows:

1. Find noes in a specific range;
2. for each node, find closes ancestor;
3. if no link is present, create a virtual like between node and ancestor;
4. connect events to visible components, from the specified time window.

Data scoping is used not just to create custom views for different role players, but to also create a scoped view of the components that are affected by a failure. In this paper, we refer to such view as a *correlated group*. The primary use of a correlated group is to be (1) a reference for other role players that are co-diagnosing the same failure or (2) a fingerprint

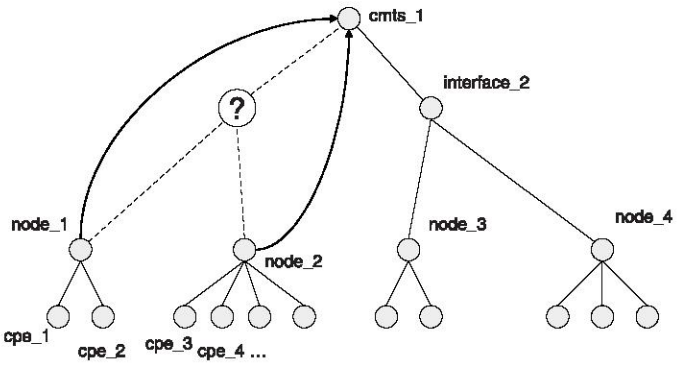

Fig. 6. Example of data scoper reconnecting missing links.

used by an automated problem management system. Consider incident management systems like ManageNow [8] or Remedy [2], where formal trouble tickets are created to dispatch work-orders to technicians. Because failures can trigger large number of alarms, which are reported by different monitoring systems as well as user calls, the generated tickets would refer to one or more of the effected components as textual input in the ticket. To help bridge the gap between process-based solutions (like incident management systems) and an IT based solution (like 3i) and to maximize the efficiency of the governing processes, correlated groups can be pushed automatically (via a SOAP interface) into incident management systems. They can be queried (as if they are a single component) for their aggregate status (e.g., to allow a technician to check if a fix has resolved all of the corresponding alarms).

B. Aggregation and Status Projection

From a user's perspective, the map contains a set of components (e.g., CMTS, interfaces, cable modems, etc), each with specific event information. However, a typical map cannot display a large number of components, especially for lower-level ones (like cable modems). For example, a city may contain hundreds of thousands of homes, each with multiple monitored devices (e.g., one or more set-top boxes and/or a cable modem). It is natural then that at a high-level view, an aggregate status of the low-level components is presented. This aggregate is based on aggregation functions that operates on the spatial information, not the logical one.

As shown in the example of Figure 7, two specific issues arise. First, aggregation does not necessarily reflect logical groupings. That is, if one is drawing a grid or a shaded box, that box might span multiple branches of the monitored graph. Second, aggregation can only reflect information about a single component layer. For example, it does not make sense to aggregate the status of fiber nodes and cable modems in Figure 7. To that extent, we define the spatial aggregation function for a given geographical box (defined using longitudes and latitudes) as follows:

$$s_{box} = \sum_{x \in C_l(box)} s_x(l) \qquad (2)$$

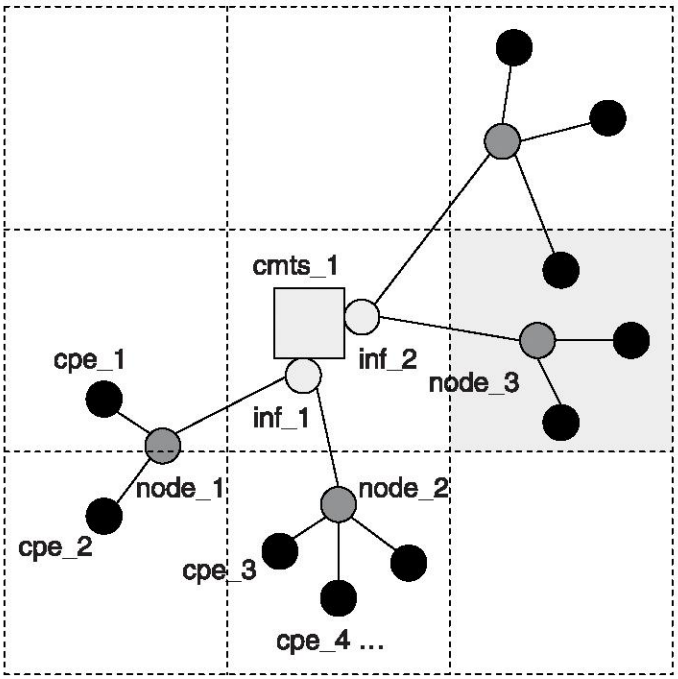

Fig. 7. Aggregation by spatial relationship

where $C_l(box)$ represents the components at topological level l in the corresponding bounding box, and s_x is the status of component x. We also assume that $\sum$ reflects an appropriate aggregation algebra of two component statuses (e.g., sum of a red and a green alarm is red). For instance, if we consider *customer premise equipments* (CPEs), $l = 0$ as depicted in Figure 5. Then, for a given area, $C_0(box)$ would contain all CPEs in that area and $s_x(0)$ corresponds to the status of a single CPE.

Notice here that we have defined s_x as a also function of l, which is the *projection level*. The projection level reflects a topological level, and $s_x(l)$ is then the status of the ancestor of component x at level l. The idea is that a device can inherit (during the computation of the status) the status of one of its parents.

Clearly, status projection can be used to quickly identify impacted areas by a specific failure. For example, by projecting the status of an alarm in the CMTS level onto the CPE components, the technician can quickly visualize the impacted area. The impacted area is the area covered by leaf nodes, like CPEs. As we will describe in Section III-C, we have found that status projection can also improve the efficiency of active probing. In some cases, we have also found that status projection is also an effective mechanism for visual identification of misconfigured devices. Consider, for example, that device *cpe_3* in Figure 7 is connected to the wrong node (*node_1* instead of *node_2*). Then, if we project node status on cable modems, when *node_1* fails, *cpe_3* may show up in the aggregation grid as an incorrect color and can be a strong visual indicator of a failed device.

Implementing status projection can be done efficiently by caching the relationship of the parent tree for all leaf nodes. For instance, if we want to project the status of a CMTS onto customer cable modems, then for each customer, we have to identify its parent CMTS. This can be easily done offline. Then, when geographical-based aggregation is performed (as part of showing the impacted area), the status of the parent CMTS is looked up. Because this status is shared by many customers, having a small dynamic cache of this information further reduces the projection overhead.

C. Active Probing

In Figure II, we show that active probing is manually invoked to check the status of a device. We have automated this task by invoking these probes for all failed components at the end of every monitoring cycle. As expected, we have found that in many cases not all of the actively probed information is needed. The reason being that not all alarms are actively diagnosed within a probing cycle. To minimize the number of active probes, we use spatial scoping to invoke those probes on components that are in view.

Unfortunately, one cannot simply invoke active probes on components in view because at the highest-map level (e.g., city view) all of the devices can be in view—also noting that high-level view is typically the most natural starting point of infrastructure monitoring.

Here, we see another benefit of status projection, whereby at the highest levels, the status of the leaf nodes inherits the closest ancestor that is being passively probed. In the case of the *current* process of Figure II, a CPE (defined as the most granular information in p_4) would inherit the status of its immediate parent. Only when the view is scoped down that the active probe is invoked.

Consider the example of multiple interfaces reporting different failures. By projecting the status of these interfaces onto the corresponding CPEs, the technician can immediately see the possible impacted area by these failed interfaces. However, not all failure indicators are critical. For instance, a low signal-to-noise ratio, even though it can generate an alarm, it may not disrupt service. Also, if a field technician has already been dispatch to investigate that root-cause of the alarm, the technician does not need to continue looking at the alarm. A technician will prioritize which alarms to investigate first. Thus, as the technician zooms in, the probe is automatically launched for the chosen interface.

D. Discussion

In the discussion so far, we have assumed that the end-to-end graph is constructed from different applications without concern for time synchronization issues. In reality, time synchronization plays an important role in problem diagnosis. For instance, if events from application A arrive at time $t = 0, 4, 8, ...min$ and application B at times $t = 1, 3, 5, ...min$, then if at $t = 4$ an alarm is triggered, the technician must wait for another minute $t = 5$ to correlate information coming from different apps. Even worse, because different event sources (e.g., applications A and B above) may report

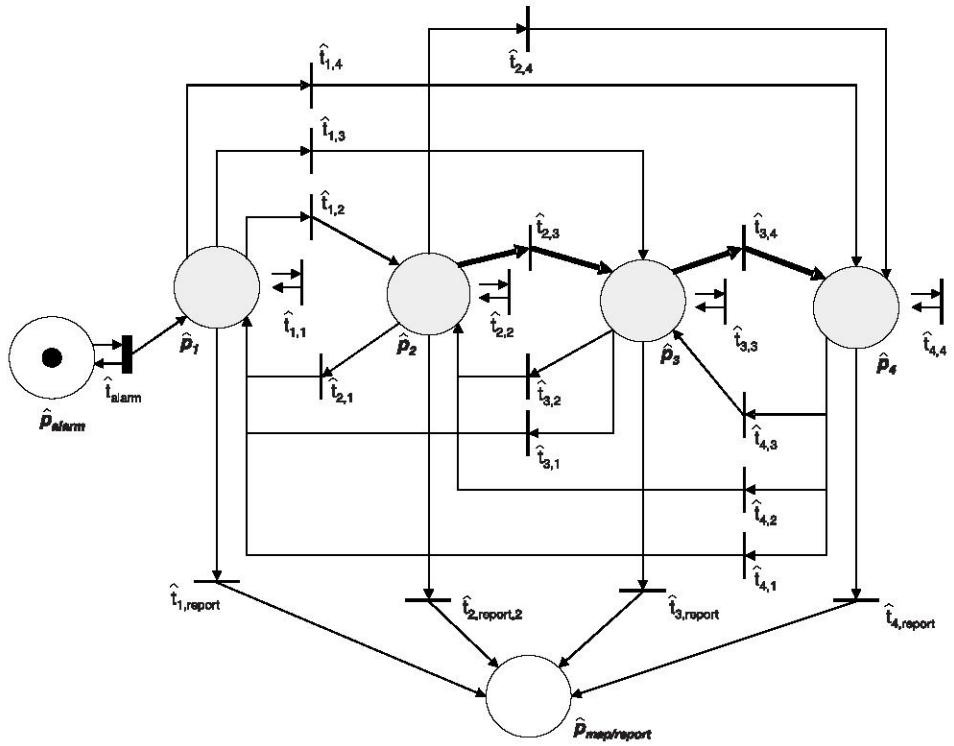

Fig. 8. Stochastic Petri-Net representation of *new* alarm detection process

alarms from different devices, a cyclical dependency between information from both applications is formed. Breaking this dependency requires better configuration of applications. In the absence of such careful application timing, we have found that contexualizing events on a map reduces some of the dependencies on information from different applications as spatial localization produces another "hint" of the source of the problem. Automating the reduction of cyclical dependencies as well as maximizing the use of spatial information is the focus of ongoing research.

IV. Evaluating the After Process

This section describes the resulting process as well as the preliminary work in evaluating the overall improvement between the old and the new ADSA process. We base our evaluation on the Process Complexity Model [4]. The model consists of a framework that approximates the complexity of IT processes. It breaks down processes into tasks, roles, and interactions, and assigns a complexity score for each task. The resulting score is a scalar value that is indicative of the total complexity of the process. By itself, the complexity score is not immediately usable. It can be used however in two ways: (1) to analyze the individual tasks that compose a process and identify bottlenecks in these tasks (i.e., quickly identify tasks with high complexity scores), and (2) to compare the absolute complexity scores of two processes and based on that derive an improvement score.

Very briefly, the model works as follows: Starting from a model of the process, identify the tasks that compose the process, the actors and their roles (the different people involved), and the data artifacts that need to be exchanged in order to perform each task. The model then tries to assign complexity metrics in the following dimensions:

- *Execution Complexity* covers the complexity involved in performing the tasks that make up the IT process. There are two metrics for execution complexity: *Base Execution Complexity* and *Decision Complexity*. The base Execution Complexity indicates complexity of the task according to its execution type. Values for this score are assigned according to a weighting scale for different task types. The Base Execution Complexity for that task is then the sum of values from all the roles. The Decision Complexity quantifies additional execution complexity due to decision making. For a non-decision making task, its value is zero. If a decision needs to be made, its complexity is based on the four complexity sub-metrics: the number of branches in the decision, the degree of supplied guidance, the consequence of impact, and the visibility of impact.
- *Coordination Complexity* measures the complexity of the different roles coordinating between the different tasks that compose the process. The coordination complexity is zero if it is linked to an automated task, and progressively increases depending on whether the artifacts transfered between the tasks need to be interpreted further.
- *Business Item Complexity* captures the complexity of involving business items, such as supplying the value of a field of a configuration item. Again, the model assigns

an increasing complexity score based on whether determining the value is automatic, or requires consideration of external information sources.

Not surprisingly, the resulting process (herein referred to as the *after* process) from integrating 3i into ADSA process enables higher efficiency. Informally, higher efficiency is achieved across two complexity parameters.

- First, the consolidation of alarm sources yields improvements in the Base Execution Complexity by reducing the number of application monitoring tasks to one, and the Decision Complexity is reduced to zero since there are no decision branches.
- Second, the drill-down process also reduces the number of tasks. This again impacts both the Base Execution Complexity and the Decision Complexity. The Coordination Complexity is also reduced as a result of the previous reduction in the number of tasks.

The resulting SPN process is depicted in Figure III-C. Unlike the before process, the resulting process can be divided into three groups: (1) alarm generation, (2) process drill-down, and (3) reporting, with the cross-referencing group eliminated.

A. Alarm Generation

In the after process, there is only a single and unified event source of alarms. This source represent events from all passive and active probes (p_{alarm} and $p_{active-probe}$ in the before process). This had a clear impact on the day-to-day operation as it eliminated the need for a technician to monitor the disparate applications for status changes. The improvement can be observed by looking at the SPN, specifically between $\hat{p}_3$ and $\hat{p}_4$. This improvement is due to two reasons: (1) elimination of transition delay $t_{active-probe}$, this effectively reduces the wait time to get the results of the active probe, and (2) consolidation of $p_{active-probe}$ as described in Section III-C. Looking carefully at the before and after processes, quantifying the improvement is not straightforward because the before process allows for the concurrent invocation (hence, pipelining) of active probes.

B. Process Drill-Down

The drill-down process is more connected and has smaller transition weights. The former is a result of two features: (1) dynamic projection and (2) event scoping. As we described earlier event scoping allows a role player to customize the monitored event graph. Specifically, by allowing a role-player to select what components to monitor, the corresponding number of drill-down steps also changes. For example, if a role player chooses to monitor the CMTS and CPE devices only in Figure 5, then the resulting process will only have two steps in the drill-down process. To capture the variation in role-player preference, we add transitions between every pair in the drill-down process.

In Section II, we mention that the transition weights are a function of the branching factor and failure probabilities of devices at each hierarchical level. Because monitoring applications traditionally use list views[4], drilling down typically requires investigating different failures for each entry of the view. Using a map, on the other hand, enables a role player to see not just a large set of components, but also provide additional geographical context. This enhanced context, from our experience, improved fault localization and verification. Both reduce the effect transition weight. Additionally, status projection further improves the drill-down requirement as it allows a role player to immediately identify impacted areas.

Consider the example in Section II where a sensor for a specific interface generates an alarm. At this point, the technician would project the status of the interface on the customer base to visualize the impacted area. The role player can also project the status of the underlying fiber nodes to quickly see if any of them is the source of the alarm. In either case (a fiber node is the cause of the alarm or not), the technician can zoom to the affected area (automatically launching the active probe) to view street-level information.

The example above brings us to the need to capture the map navigation in the after process. These are captured by transitions $\hat{t}_{1,1}$, $\hat{t}_{2,2}$, $\hat{t}_{3,3}$ and $\hat{t}_{4,4}$. These transitions have a weight of 1 as map navigation does not produce additional tokens within any process step. They also point back to the same process step since navigation maintains the same context within a hierarchical level.

C. Reporting

In the after process, reporting includes both geographical and logical information and can be produces at all process steps. Because these reports also cover impacted areas, they depict the customers that are experiencing failures and the ones that do not. Not surprisingly, we found that for problems like cable cuts, where a large number of geographically proximate customers are affected, such reports can pinpoint good starting locations to field technicians (e.g., between homes showing green status and those showing red).

V. Related Work

The topic of infrastructure monitoring and management has been explored in many research studies (e.g., [7, 12, 14, 15, 19, 21], to name a few). It has also been the focus of commercial and open source tools [5, 17]. In this section, we will look at a narrower scope. Specifically, we will look at related work that intersects with location-aware or GIS-based technologies.

OpenGIS defined various standards for encoding and dealing with observations and measurements [18]. While we present monitored events in the backdrop of a geo-coded map, the scope of OpenGIS is much broader, as it defines inter-operable GIS-based applications. At the same time, the OpenGIS standard does not address specific problems in service assurance process that we have described. Indulska and Orlowska [10] have looked at aggregation issues in spatial data management, focusing on database data aggregation. Their

[4] a *list view* is used here to refer to tables or tree branches that list a set of components, each with some status information

scheme focuses on information pre-processing to improve the efficiency of the back-end data-store. In 3i, we address the data aggregation by restricting frequent spatial aggregation queries to a fixed grid (with different granularity at different zoom levels). This allowed us to efficiently pre-compute the aggregates. 3i also supports more dynamic spatial aggregation. These are less frequently accessed (once every several minutes) and incur, on average, a 3 second delay.

In [23], the authors investigate scripting in GIS systems. Large portions (specifically, application server components) of 3i is implemented in PHP [20], which, can thus be viewed as an example of the usefulness of scripted languages to create advanced monitoring solutions. The authors of [1] present a specification language for describing events that can be used in location-aware systems. Our work, encompasses a similar approach during the normalization of events. We have focused on using such event information to improve the efficiency of the corresponding management process. In [3], the authors look at using spatial information to customize information being published to mobile users. While this seems a departure from the focus of this paper, our approach of using spatial information to launch an active probe is related to some extent. Here, we are using the view of a technician as an indicator of what information to subscribe to.

From a tooling perspective, solutions like iGlass [9] provide a location-aware view of the cable infrastructure. Our work in this paper was not just to implement a location-aware system, but to address key performance and implementation issues that affects in managing a large number of interrelated components, thus ultimately, affecting the corresponding service assurance process.

VI. Conclusions

GIS technology has the potential of trasforming many aspects of the service management process in converged IP networks. Based on an engagement with a large Cable provider, we have designed and built 3i to address key issues in the service assurance process. 3i is highly componentized and provides an intuitive way for creating role-based views through dynamic scoping, event aggregation and status projection, and location-driven active probing. We analyzed the current service assurance process and the improvements after introducing 3i. Beyond the service assurance process, we are already seeing industry interest in using it for planning and marketing, since it can study changes in customer densities and underlying infrastructure capacities and correlate with growth numbers in new service offerings. We have observed similar interest from other service providers in the Telco and Cable space. It is also applicable to the utilities industry. We have a fully functional prototype working on live operations data.

We are actively working on quantifying the productivity improvements of using 3i by modeling in detail the before and after process using a process modeling framework. We anticipate to present these results in an extended version of this paper.

References

[1] M. Bauer and K. Rothermel, "Towards the Observation of Spatial Events in Distributed Location-Aware Systems," in *ICDCSW '02: Proceedings of the 22nd International Conference on Distributed Computing Systems*. Washington, DC, USA: IEEE Computer Society, 2002, pp. 581–582.

[2] BMC Software, "BMC Remedy IT Service Management," http://www.bmc.com.

[3] X. Chen, Y. Chen, and F. Rao, "An Efficient Spatial Publish/Subscribe System for Intelligent Location-based Services," in *DEBS '03: Proceedings of the 2nd international workshop on Distributed event-based systems*. New York, NY, USA: ACM Press, 2003, pp. 1–6.

[4] Y. Diao and A. Keller, "Quantifying the Complexity of IT Service Management Processes," in *Proceedings of the 17th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management*, Dublin, Ireland, October 2006.

[5] EMC, "SMARTS," http://www.emc.com.

[6] Google, "Google Maps API," http://www.google.com/apis/maps.

[7] K. Houck, S. Calo, and A. Finkel, "Towards a practical alarm correlation system," in *Proceedings of the fourth international symposium on Integrated network management IV*. London, UK, UK: Chapman & Hall, Ltd., 1995, pp. 226–237.

[8] IBM Corp., "ManageNow," http://www.ibm.com.

[9] iGlass, "iGlass," http://www.iglass.net.

[10] M. Indulska and M. E. Orlowska, "On Aggregation Issues in Spatial Data Management," in *ADC '02: Proceedings of the thirteenth Australasian database conference*. Darlinghurst, Australia, Australia: Australian Computer Society, Inc., 2002, pp. 75–84.

[11] H. Jamjoom, N. Anerousis, and D. Saha, "Integrated Infrastructure Intelligence (an Overview)," 2006, http://domino.research.ibm.com/comm/research_projects.nsf /pages/i3.index.html.

[12] K. Julisch, "Mining Alarm Clusters to Improve Alarm Handling Efficiency," in *ACSAC '01: Proceedings of the 17th Annual Computer Security Applications Conference*. Washington, DC, USA: IEEE Computer Society, 2001, p. 12.

[13] M. B. Kelly, "The TeleManagement Forum's Enhanced Telecom Operations Map (eTOM)," *Journal of Network System Managment*, vol. 11, no. 1, 2003.

[14] S. Klinger, S. Yemini, Y. Yemini, D. Ohsie, and S. Stolfo, "A Coding Approach to Event Correlation," in *Proceedings of the Fourth International Symposium on Integrated Network Management IV*. London, UK, UK: Chapman & Hall, Ltd., 1995, pp. 266–277.

[15] S. Ma and J. L. Hellerstein, "EventBrowser: A Flexible Tool for Scalable Analysis of Event Data," in *DSOM '99: Proceedings of the 10th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management*. London, UK: Springer-Verlag, 1999, pp. 285–296.

[16] M. A. Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis, *Modelling with Generalized Stochastic Petri Nets*. John Wiley and Sons, 1995.

[17] Micromuse, "Netcool," http://www.micromuse.com.

[18] Open Geospatial Consortium, Inc., "OpenGIS Specifications," http://www.opengeospatial.org/standards.

[19] C.-S. Perng, D. Thoenen, G. Grabarnik, S. Ma, and J. Hellerstein, "Data-driven Validation, Completion and Construction of Event Relationship Networks," in *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*. New York, NY, USA: ACM Press, 2003, pp. 729–734.

[20] PHP, "PHP: Hypertext Preprocessor," http://www.php.net.

[21] R. V. Renesse, K. P. Birman, and W. Vogels, "Astrolabe: A Robust and Scalable Technology for Distributed System Monitoring, Management, and Data Mining," *ACM Transactions on Computer Systems (TOCS)*, vol. 21, no. 2, pp. 164–206, May 2003.

[22] H. Samet, *The Design and Analysis of Spatial Data Structures*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1990.

[23] A. Sorokine and K. Ackermann, "Scripting in GIS Applications: Experimental Standards-based Framework for Perl," in *GIS '00: Proceedings of the 8th ACM international symposium on Advances in geographic information systems*. New York, NY, USA: ACM Press, 2000, pp. 102–107.