2012 9th International Conference on Ubiquitous Intelligence and Computing and 9th International Conference on Autonomic and Trusted Computing

Ubiquitous Smart Devices and Applications for Disaster Preparedness

W. P. Liao and Y. Z. Ou
National Tsing-Hua Univ.
Dept. of Computer Science
Hsinchu, Taiwan, {wpliao, yzou}@cs.nthu.edu.tw

E. T. H. Chu
National YunLin Tech.
Dept. of Computer Science
Yunlin, Taiwan
edwardchu@yuntech.edu.tw

C. S. Shih
National Taiwan Univ.
Dept. of CS and Info. Eng
Taipei, Taiwan
cshih@csie.ntu.edu.tw

J. W. S. Liu
Academia Sinica.
Inst. of Information Science
Taipei, Taiwan
janeliu@iis.sinica.edu.tw

***Abstract*—Recent advances in disaster prediction and detection technologies and ICT support infrastructures have enabled the generation and reliable deliveries of machine-readable early disaster alerts over all communication pathways. The emergence of ubiquitous smart devices and applications that can receive, authenticate and process standard-conforming disaster alert messages and respond by taking appropriate actions to help us to be better prepared for nature disasters is a natural next step in the advancement of disaster management technologies. We call such smart devices and applications iGaDs (intelligent Guards against Disasters). This paper describes reference architecture, key components and design of iGaDs in general and an ASIC enhancement of battery-powered iGaDs.**

Keywords-ubiquitous computing, embedded devices, smart environment, disaster preparedness

I. INTRODUCTION

In recent decades, we have seen tremendous advances in technologies for the predication and detection of nature disasters and ICT infrastructures for generation and distribution of disaster alerts/warnings. Today, most developed regions of the world are literally covered by diverse in-situ and remote sensors [1, 2], ranging from surveillance and eco sensors in oceans, to advanced weather radars, to sensors monitoring surface and underground water and road and bridge conditions, to broadband seismometer arrays and strong motion sensors, and so on. Efforts of large projects (e.g., OSIRIS [3], SANY [4] and SensorNet [5]) and standards organizations (e.g., OGC [6] and OASIS [7]) have made available standards and tools needed to support interoperability of diverse sensor networks and sensor webs and processing and use of multi-domain, real-time sensor data and information provided by them.

Recent advances in ICT infrastructures include platforms for emergency services. An example is IPAWS-OPEN (Integrated Public Alert and Warning System – Open Platform for Emergency Networks) [8] in the USA. The platform provides services to receive and authenticate standard-based messages from alerting authorities and then broadcasts the messages via all communication pathways, including digital radio broadcast, cellular networks and Internet. XML-based message format standards CAP (Common Alerting Protocol) [9] and EDXL-DE (Emergency Data Exchange Language Distribution Element) [10] enable information exchanges between emergency information systems and public safety organizations, automatic report by sensor systems to analysis centers, and aggregation and correlation of warnings from multiple sources. Consequently, alert decision support systems (e.g., [11-16]) not only can generate more accurate alerts earlier but also can have the alerts delivered sooner and used for more purposes than possible a few years ago.

This paper describes the design and implementation of smart embedded devices, systems, and applications that can receive, authenticate and process standard-based disaster alert messages and respond by taking specified actions to help us minimize personal dangers and reduce property damages and economic losses when disasters strike. We call such devices and applications *iGaDs* (*intelligent Guards against Disasters*) [17] in general. For sake of concreteness, our discussions assume that alert messages conform to CAP standard and are distributed by IPAWS, and every iGaD can receive alerts via one or more of cellular networks, Internet, and other wireless connections. We will highlight the capability of devices and applications to process and respond to CAP messages by saying that they are *CAP-aware*.

The CAP and IPAWS assumption is not as restrictive as it seems. What will be said about iGaDs in later sections are applicable even when messages do not conform to CAP, provided that they are in a XML format, are signed and broadcast by a trusted service via all communication pathways and contain information needed to support iGaDs decisions.

As examples, Fig. 1 shows two embedded iGaDs designed to respond to warnings of imminent strong earthquakes. Nowadays, earthquake-prone regions such as Taiwan, Japan and parts of USA and Mexico are monitored by broadband arrays of seismometers and strong seismic motion sensors. Seismic data sent by them via RF to computers at analysis centers enable the determination of the focus and magnitude of each earthquake, and in case of severe quakes, the broadcast of alerts a fraction of a second or more before shock waves arrive and ground motion starts in the affected areas. When warned of an earthquake of a specified magnitude or stronger, a CAP-aware elevator controller in a smart building slows down elevators and stops them when they reach the closest floors, as shown in the left half the figure.

This work was supported in part by Academia Sinica thematic project OpenISDM (Open Information Systems for Disaster Management, AS-101-TP2-A01, and by the Taiwan National Science Council Grants Nos. NSC 100-2218-E-224-001-MY2 and NSC 100-2219-E-224-001.

978-0-7695-4843-2/12 $26.00 © 2012 IEEE
DOI 10.1109/UIC-ATC.2012.98

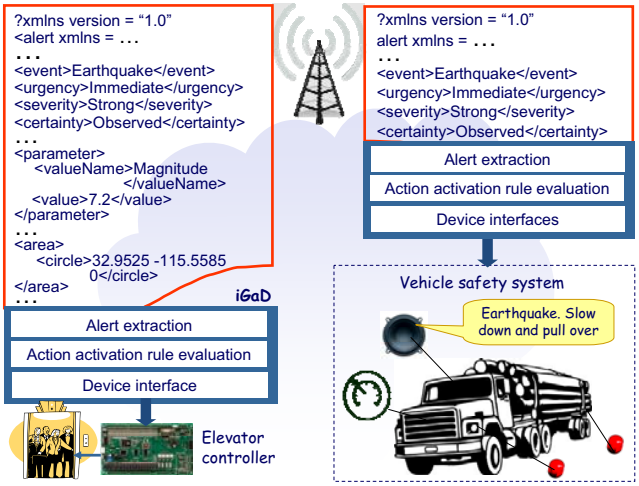

Fig. 1 Examples of embedded iGaDs

In addition to CAP-aware elevator controllers, a smart building may have iGaDs that unlock entry-access controlled doors to ease evacuation, shut down natural gas flow into the building to prevent fire, and so on. The other example shown in Fig. 1 is a CAP-aware vehicle safety system on board a truck traveling in an affected area. When triggered by a strong earthquake alarm, it warns the driver of the imminent strong earthquake, turns on the hazard flashers, disengages the cruise control and helps the driver to slow down. A CAP-aware variable message sign system broadcasts text messages to signs before tunnels and bridges on highways to tell drivers to slowdown and pull over.

In addition to embedded iGaDs, software iGaDs are CAP-aware applications that run on computers, smart phones and other platforms with sufficient computing power and memory space and some form of location-based service. Examples include applications that respond to severe earthquake alerts by telling surgeons in hospitals to pause on-going operations, or shoppers in supermarkets of relatively safe aisles to be during the quake, or children in schools to stay calm and take cover under their desks, and so on. Other examples of iGaDs for preparedness against major earthquakes and other types of natural disasters, together with discussions on how they can help, can be found in [17].

Following this introduction, Section 2 describes related works. Section 3 describes reference architecture and key components for building configurable and customizable iGaDs. Section 4 describes a design and implementation of embedded iGaDs and an ASIC enhancement for battery powered iGaDs. Section 5 summarizes the paper and discusses future works.

II. Related Works

We focus here primarily on embedded iGaDs. They can be viewed as actuators in a large, distributed cyber-physical system with diverse sensors, complex decision support servers and communication networks. Being dependable and responsive is essential. Some iGaDs, including the examples mentioned above, aim to complement and to be integrated with the increasingly broader spectrum of devices, applications and services offered by modern smart homes and environments to make the environments safer against disasters. Problems related to their integration are out of scope of this paper. We will not address them except to say that technologies and protocols for smart environments [18] should be used for this purpose.

As stated earlier, iGaDs receive and respond to alert messages in CAP [9] format sent by alert authorities via IPAWS – OPEN [8]. In this aspect, they resemble CAP-EAS decoders in the public emergency alerting systems (EAS) [19] in the USA for use by the President to address the public during national emergencies and by state and local authorities for delivery of emergency information, including severe weather and earthquake alerts, to people in specific areas. Since late 2010, FEMA has adopted CAP v1.2 and implemented IPAWS-OPEN. Tests of CAP-EAS are now being conducted regularly. Some of the implementation guidelines published recently by CAP-EAS Industry Group are applicable to the design and implementation of iGaDs, especially for iGaDs that function as emergency alert systems for large buildings, shopping malls, etc. Many common requirements (e.g., low cost and power consumption), in addition to being easily configurable and customizable to work in different environments and conditions, make the design and implementation of iGaDs uniquely challenging, however. We will return to discuss problems in these respects and describe solutions to them.

XML processing capabilities is essential since alert messages are in a XML format. Software iGaDs have a wide range of choices among matured XML parsers for operating systems and programming languages supported by popular computer and mobile platforms. In addition to the open source Java library [20] for parsing CAP messages specifically, choices of XML parsers include NSXML and libxml2 [21, 22] available for Apple iOS and SAXParser [23] in Android Java SDK. Windows Phones has LINQ to XML [24], which is an XML programming interface.

Hardware XML parsers have begun to emerge and offer us an alternative to software solutions. Existing hardware parsers (e.g., [25]) are typically designed for applications that need high throughput and processing speed. In contrast, we need lightweight parsers that can extract alert information from CAP messages within a fraction of a second with minimum energy.

As the next section will explain, iGaDs also include as a key component a rule engine. Similar to XML parsers, there are many matured software inference/rule engines. Examples include CLIPS [26] and JESS [27]; they are widely used in context-aware applications. Table-based rule engines (e.g., Logician [28]) and rule engines for mobile devices (e.g., MiRE [29]) are also sufficient for iGaDs.

Rather than using such a rule engine, Section 4 will present a lightweight scheme for processing the rules that govern the choices of actions of embedded iGaDs. iGaDs based on this scheme can be easily customized with minimal overhead.

III. Reference Architecture and Major Components

This section first discusses our design and architecture choices for all iGaDs and the rationale behind them. It then describes a general structure and major components.

A. *Design Choices and Rationale*

Despite significant differences in their functionalities, different types of iGaDs share many requirements, which include configurability, customizability and affordability. Building them from configurable components within a general architectural framework is a try-and-true way to make them affordable. Configurability of iGaDs goes beyond this level, however: Individual iGaDs of the same type are often required to respond to the same alert differently, depending on where and how they are used.

To illustrate, let us consider an iGaD guarding an entry-access controlled door of an office building that is a designated public severe storm shelter. The door is normally locked. When alerted of an EF (Enhanced Fujita) 4 or 5 tornado, the iGaD is required to respond by unlocking the door in order to make the building accessible to all people. The iGaD also unlocks the door in response to alerts of earthquakes magnitude 8 or stronger to ease the evacuation of people in the building. In other words, the iGaD shall send an unlock command to the building door in response to an alert message if either one or both of the following rules is true.

(1) `(EventType == "Earthquake") AND (Scale >= THRESHOLD_MAGNITUDE)`

(2) `(EventType == "Tornado") AND (Scale >= THRESHOLD_SEVERITY)`

The values of variables EventType and Scale in the expressions of these rules are the standard name of the event (e.g., earthquake, tornado, tropical cyclone, etc.) and the value of the severity measure (e.g., in the USA, Modified Mercalli (MM) intensity scale for earthquakes, Enhanced Fujita (EF) scale for strength of tornados, and category for hurricanes, etc.) specified by the alert. They are extracted from the message by an XML parser before the rules are evaluated. Parameters with all capital-letter names are configuration parameters of the iGaD. Here, THRESHOLD_MAGNITUDE is the maximum earthquake magnitude the building is constructed to withstand, and THRESHOLD_SEVERITY is the minimum severity of tornados for which the building is used as a public shelter. By setting these parameters at installation time and maintenance time, we can customize the iGaD according to the construction, condition and usage of the building.

We call rules such as (1) and (2) *action activation rules* of the iGaD. In addition to changing configuration parameters that define the rules, action decisions of individual iGaDs of the same type can also be customized by providing them with individualized rules. As an example, suppose that an iGaD for outside doors of a smart home is the same as the iGaD for a shelter door. This iGaD should also command the doors to open in response to a strong earthquake, though likely of a lower threshold scale. During a tornado emergency, however, the outside doors and vents of the house should open in order to equalize air pressures in and out of the house, but only if and when a tornado actually hits the house. We can take into account of this consideration by letting the iGaD have an additional rule defined in terms of the readings of digital barometers inside and outside of the house. Rule (3) stated below is an example: The iGaD sends open commands to the outside doors when rule (1) is true or when rules (2) and (3) are both true.

(3) `InsideAirPressure >= OutsideAirPressure * THRESHOLD_RATIO`

These examples motivated us to use a rule engine to provide iGaDs with decision support. The action activation rules of typical iGaDs can be expressed in terms of propositional logic or predicate logic, simple enough to be evaluated as described in Section 4.

B. *Major Components*

Fig. 2 shows the major functional components of embedded iGaDs and information flows among them. We can logically partition an embedded iGaD into a CAP message processor and a device controller. A software iGaD does not have a device controller; a CAP-aware application takes its place.

The CAP message processor is essential the same for all iGaDs. It is responsible for extracting from each alert message the information needed by the iGaD device controller (or application) to decide whether and how to respond.

In addition to making decisions regarding actions taken by the iGaD, the device controller controls one or more physical device(s) (e.g., automatic lock and gas valve) connected physically to it or via one or more networks. The interface with each device clearly depends on the device. Specific details in their interconnections and interactions are unimportant for the sake of our discussion here. It suffices to say that the iGaD interacts with the driver of each physical device via events: The device driver handles events from the iGaD as commands (e.g., lock and unlock commands), and events from the drivers are treated by the iGaD as acknowledgements.

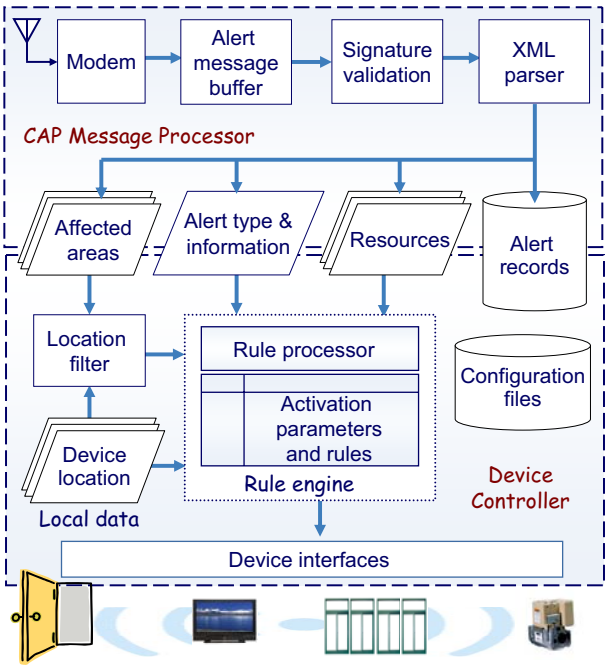

Fig. 2 General structure and major components

Some iGaDs also relies on local data (e.g., device location, local sensor data, etc.) provided by its operating environment for this purpose, as shown by Fig. 2. The figure omits interfaces to them as well.

As Fig. 2 indicates, once a message arrives in the input message buffer, it is authenticated by the signature validation module based on its enveloped digital signature. The module also stores and maintains the key(s) in ways dictated by the message standard. Again, the information extracted by the XML parser from each alert message received by the iGaD include the name and scale of the type and severity of the alert event, as well as specifications of areas affected by the alert. The message may also provide resources such as human-readable descriptions and URLs of files containing supplement information (e.g., photos, maps, audio, and so on) that may be useful to the public EAS and some iGaDs.

The C-like data structures in Fig. 3 illustrate how iGaD maintains the output of the XML parser. BasicAlertInfo holds values of EventType and Scale and other values extracted from the beginning of the CAP message. These values enable the iGaD to *screen* the message quickly to determine whether the alert event described by the message is of one of its *attend-to* types (i.e., event types for which the iGaD is designed to respond) and whether the event is of a scale warranting its further attention. Message screening is important for typical embedded iGaDs that have only one or a few attend-to event types. The next section will discuss this issue further.

The XML parser processes the remaining part of the message and saves the extracted values in AdditionalAlertInfo only when the message passes message screening. Once the iGaD has ascertained that the alert event is of an attend-to type and with severity within the specified scale range of the type, the location filter is invoked to determine whether any of the physical devices controlled by the iGaDs is located in an affected area defined by a polygon, or a circle, or a geocode contained in the alert element and hence is targeted by the alert. If any of them is located in an affected area, the iGaD then evaluates action activation rules and issues commands accordingly to physical devices controlled by it.

As stated earlier, the alert message may include additional parameters of the event, expiration times, image and text descriptions, and so on. Some iGaDs (e.g., those that provide emergency alert functions of building management systems) can use them as suggested by the CAP-EAS implementation guidelines [19] in the generation of their responses.

IV. Design and Implementtion

This section first describes a design of embedded iGaDs. The design aims to make them highly configurable and customizable while keeping their processing and memory usages small. There are good reasons to off-load some or all of the CAP-awareness functions to an ASIC (Application-Specific Integrated Circuit) or a microcontroller. The section presents the reason for off-loading the message screening function to an ASIC as a minimal hardware enhancement of battery powered iGaDs and then describes a proof of concept ASIC prototype.

```
typedef struct BasicAlertInfo  {
  Identifier;     // Message identifier
  …   // Sender ID, References, Scope, etc.
  Status;        // Actual or Exercise
  MsgType;   // Alert, Cancel, Error, .
  Scope;         //.Public, Restricted, Private
  Category;    // Geo, Met, Health, etc.
  EventType; // Earthquake, Tornado, etc.
  Scale;         // Value of severity measure
  Urgency;     // Immediate, Future, etc.
  Certainty;    // Observed, Likely, etc.
  …
  * AdditionalAlertInfo;
} BASIC_ALERT_INFO;

typedef struct AffectArea  {
   union {
     struct PolygonCoordinates;
     struct CircleCenterRadius;
     GeoCode;
     AffectedAreaListEntry; }
} AFFECTED_AREA
typedef struct AdditionalAlertInfo  {
   * BasicAlertInfo;
   AffectedAreaListHead;
   ParameterListHead;
   ResourceListHead;
   … // Description, instructions, etc.
} ADDITIONAL_ALERT_INFO
```

Fig. 3 Basic and addttional alert information

A. A Design Pattern for Configurable iGaDs

Fig. 4 describes in pseudo code a design for configurable iGaDs. To keep the description simple, all error handling paths are omitted. The figure also omits specifics on the interfaces between the iGaD and the physical devices controlled by it and iGaD and local sensors and services relied on by it.

This version uses a message handler (MsgHandler) thread to stream input messages to one or more alert message buffers. The remaining functions of the iGaD are performed by the main thread.

```
Major data structures
typedef struct PhysDevInterfaces {
  Information on physical devices;
  Events for interaction with drivers;
  …
} PHYS _DEV_INTERFACES;

typedef struct iGaDLocation {
   Coordinates; Geocode;
} iGaD_LOCATION;

typedef struct LocalData {
   …      // Other local data
} LOCAL_DATA;

typedef struct iGaDConfig {
   SizeOfNamesScalesRules;
   NoAttendToTypes;
   NoRuleFunctions;
   TotalSizeOfRuleFunctions;
   Other information on rule library
      for loading rule functions;
} iGAD_CONFIG;

typedef struct EventConfig {
    TypeName;
    ScaleUpperThreshold;
    ScaleLowerThreshold;
    * ActivationRuleFunc;
} EVENT_CONFIG;

// Variables shared with MsgHandler
typedef struct SharedVar {
   ReadyEvent;  ExitEvent;
   MessageWaitingEvent;
   ProcessingDoneEvent;
   * AlertMessageBuffer;
} SHARED_VAR;

Void main {  // iGad main thread
   iGaDLocation; localData;
   iGaDConfig; SharedVar;
   PhysDevInterfaces; AlertMessageBuffer;
   BasicEventInfo; AdditionalEventInfo;
   EventConfig[ ];
   Initialize PhysDevInterfaces;
   Copy ConfigParameters file to iGaDConfig;
         // iGaDConfig is initialize
   EventConfig = calloc (
         iGaDConfig.SizeOfNamesScalesRules);
   Copy NamesScalesRules file to the pool;
         // EventConfig array is initialized
   Complete load rule functions;
   Initialize events and complete initialization;
   Create MsgHandler thread;
   Set SharedVar.ReadyEvent;
W: Waitfor SharedVar.MessageWaitingEvent;
   // Screen the message
   XMLParser (* BasicEventInfo,
                AlertMessageBuffer);
   for every element i of  EventConfig[ ] {
     if (current event is of attend-to type & its
       scale is in threshold range of the type) {
       // The message passed.
       XMLParser (* AdditionalEventInfo,
                    AlertMessageBuffer);
       if (DeviceLocation is an affected area
                  listed in AddditionalEventInfo) {
         invoke ActivationRuleFunc of the type;
         if (Some action is to be taken)
            Raise events in PhyDevInterfaces;
         break;
       } // iGad not in an affected area,
     } // Continue to search EventConfig array
   } // The event can be ignored
   Log alert; Set ProcessingDone;
   goto W; // go to wait for new message
   …       // Run until shutdown
   set SharedVar.ExitEvent;
   wait until MsgHandler exits;
   cleanup and return;
}
```

Fig. 4 Pseudo code description of embedded iGaDs operatons

In general, the number of message handler threads and numbers of threads for XML parsing and rule evaluation are configuration parameters of the iGaD. The configuration described here is appropriate for simple iGaDs that have only a small number of attend-to event types. We have seen examples earlier. iGaDs that are required to respond to numerous alert types in ways governed by numerous and complex rules may need to use multiple threads to manage message buffers and parse messages and processes action activation rules.

The threads in an iGaD communicate via events: After authenticating the message in the current buffer, MsgHandler sets MessageWaitingEvent to indicate that the message is ready to be processed. ProcessingDoneEvent is set by the main thread to let MsgHandler know that the message in the buffer can be discarded. In addition, the main thread sets ReadyEvent and ExitEvent to tell MsgHandler that the iGaD is ready to work and wants to terminate, respectively.

Another design choice of this version is to use a library of (action activation) rule functions to support action activation decisions of the iGaD. The library is loaded at run time into the address space of the iGaD executable. This choice is also for simple iGaDs that have a few rules. (Without loss of generality, the description assumes one rule function per attend-to event type.) For iGaDs with numerous or complex rules, a better alternative is to use a general-purpose rule engine (e.g., [28, 29]) capable of processing all the rules of all kinds of iGaDs.

The left half of Fig. 4 lists the major data structures. Some of the configuration parameters of the iGaD are held in iGaDConfig, a structure of known size. It is initialized by copying the values of an identical structure stored in the configuration file ConfigParameters. Elements of iGaDConfig provide the iGaD thread with the sizes of remaining configuration data and the rule library so that memory can be allocated for them dynamically, as indicated in the lines in the top-right part of Fig. 4.

An array of EventConfig structures, one for each attend-to event type of the iGaD, holds other configuration parameters: Specifically, each EventConfig holds the name, scale thresholds and a pointer to the rule function for an attend-to event type. During initialization, the values of these structures, as well as executables of the rule functions, are copied from the configuration file NamesScalesRules. In short, the iGaD thread initializes every data structure or function that needs to be customized for the iGaD by copying the corresponding structure or function stored in its configuration files. Finally, as its name indicates, SharedVar structure holds variables shared by the threads.

Lines in the right column in Fig. 4 describe the work done by the main thread: After it successfully completes initialization, it waits for MessageWaitingEvent to be set by MsgHandler. When awaken, the thread first screens the message: It extracts from the message the event type and severity of the alert and compares them with the event type and scale thresholds held in EventConfig[ ] to determine whether the alert is of the type and severity for which it is required to respond. If the message passes the screening test, the main thread proceeds to parse the rest of the message and determine whether and how to respond to the alert. This work is done by the for loop listed in the bottom half of the right column.

Clearly, this and similar designs can be easily implemented to run on common computing and mobile device platforms. Modern low-cost, low-power microcontrollers (e.g., the 32-bit Stellaris LM3S1016) capable of delivering tens of Dhrystone MIPS at less than 5 mW per MIPS are well suited for embedded iGaDs.

We can also argue for the merits of using XML parsers and rule engines (e.g. [21-29]) now available on popular platforms to implement CAP-awareness functions of iGaD applications that run on smart phones, PDAs, and computers. In a browser-based programming environment, we also can use a JAVA script to extract alert information and process action activation rules. Indeed, these options are all reasonable for iGaDs (such as CAP-aware elevator controllers and vehicle and building safety systems) that are always powered on and plugged to uninterruptable power sources.

B. Hardware Enhancement

Pure software implementation is problematic for battery powered embedded iGaDs or CAP-aware applications on portable platforms, however. To see why, let us look at a laptop or smart phone hosting an early earthquake warning application. With no other CAP-aware applications on the platform, CAP messages other than strong earthquake alerts can be ignored. Nevertheless, all messages must be processed as soon as they arrive to see whether they are about earthquakes, even when the platform is off or in a power-saving mode. Turning the platform on or waking it up is not a problem: The laptop can be powered up in ways similar to how NOAA weather radios are turned on by alerts, and the phone can be woken up if alert messages are treated as incoming calls. However, letting all CAP messages wake up the platform is clearly not acceptable. Even in an earthquake prone region, there are only a few strong earthquake alerts per year, but probably hundreds and thousands of broadcast alert messages for all event types.

An ideal solution is to incorporate into popular platforms of computers and personal communication devices iGaD hardware components that can provide applications on the platforms with CAP-awareness capabilities. Being a XML document one to a few thousand characters in length, a typical message can be processed by the components within a second or two even when the components operate at sleep-mode clock rates of the platforms. Moreover, their cost can be minuscule if they conform to international standards such as CAP and hence, can be mass produced like ICs for phones.

The ideal solution is not going to happen until diverse iGaDs have been proven cost-effectiveness beyond doubts as tools for disaster preparedness and they are used pervasively. Until then, an ASIC message screener shown in Fig. 5 is a solution. It is primarily for iGaDs that run on battery powered platforms: It screens incoming CAP messages. When it finds a message that may require the response of one or more iGaD served by it, it wakes or powers up the platform if the platform is sleeping or off and notifies the iGaD(s) to respond.

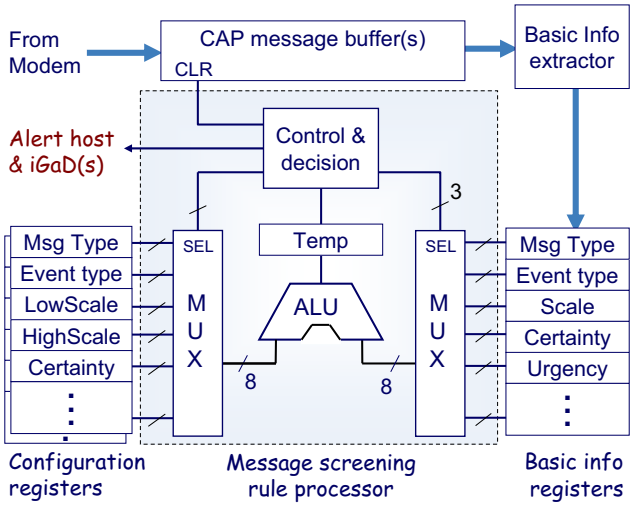

Fig. 5 ASIC CAP message screener

In the proof-of-concept message screener, basic Info extractor is a small hardware parser. It recognizes only the tags marking message type, event type, scale, urgency and certainty elements in the beginning of CAP messages. (Fig. 3 gives examples of their possible values.) After scanning the beginning of the current message in the message buffer, it extracts the values of these elements, translates the values into internal codes and places the code in the alert register file. The number of alert registers and the number of bits per register are small. (Fig. 5 shows eight registers with 8 bits per register.)

The ASIC also has a register file of similar sizes for each iGaD served it; the registers in the file hold the corresponding configuration parameters of the iGaD. Having extracted basic information from the message, the extractor enables the message screening rule processor to compare the extracted values against the corresponding configuration parameters of the iGaDs stored in their configuration registers.

For sake of simplicity, Fig. 5 does not show the clock and most of the enable lines. It shows only select lines to the multiplex, clear to the message buffer and asserts to the host and iGaD from the control and decision circuit. The circuit sequences the comparisons of register contents. If the result of a microinstruction in the sequence indicates that the message is not intended for any iGaD served by the screener, the circuit terminates the processing and clears the current message buffer. If the extracted values pass all the comparisons and hence match the corresponding configuration parameters of an iGaD, the circuit alerts the iGaD, having the platform powered or woken up as needed.

As shown here, the message screener does no other work. All the works, including authenticating the message, checking affected areas, etc. are done by the iGaD(s) once the screener finds that the message warrants attention.

Even with the help of the ASIC message screener, state-of-the-art notebooks and laptops remain to be less than ideal platforms for some iGaD applications. The reason is that they typically take 10's of second to wake up from deep sleep mode and even longer to power up. This amount of delay is too long for early earthquake alert applications used in areas such as Taiwan and parts of California where the time between the detection of an earthquake to time when earth movements are felt can be as short as a small fraction of a second. On the other hand, the delay is tolerable for storm and flood warning applications as long as it is less than a minute.

V. Summary and Future Work

In previous sections, we first discussed how iGaDs can help us be better prepare against natural disasters. The acronym iGaD stands for intelligent guards against disasters. They are embedded devices, systems of devices, and applications designed to process and respond to disaster alert messages that are in a standard XML format, generated by registered alert agencies and emergency alert services, and broadcast via all communication pathways. In addition to their being machine-readable and automatically authenticable, these alert messages differ from tweets, RTM (right this moment) and other kinds of eyewitness reports sent by crowd via social media in yet another important way: Eyewitness reports have proven to be effective tools for people to inform people of on-going emergencies. In contrast, alert messages used by iGaDs provide early warnings of imminent calamities, typically prior to their ill effects are felt by people in affected area. iGaDs aim to make use of the small lead time to prepare our living environment and ourselves in face of potential danger.

We also presented in previous sections an architectural framework for building diverse iGaDs that can be customized individually at installation and maintenance times according to their usages and operating environments. By adjusting their configuration parameters and component choices, embedded iGaDs for smart homes, smart work places, shopping malls, etc. running on computing platforms and microcontrollers that are powered up and connected to uninterruptable power sources can be implemented as suggested by the design pattern described in Section IV.

iGads running on battery powered platforms need to be enhanced by a hardware message screener such as the ASIC screener described in Section IV. The device processes incoming messages to filter out all but the small percentage of them that require the attention of the iGaDs served by it. We are experimenting with proof-of-concept prototypes of message screener and embedded iGaDs that control simple physical devices such as automatic locks and spot lights used for public shelter doors, hazard flashers in cars, etc. We are yet to assess their performance in terms of response times versus processing, memory usage and power usage overheads.

A major emphasis of our future work is on dependability: Many iGaDs are safety-critical devices or components of safety equipment. They must be highly dependable. The pseudo code description presented in Fig. 4 leaves out all the error handling paths, which normally amounts to multiple times the size of working code. Similar, rules governing actions to take when alert events are cancelled and physical device malfunctions, as well as rules for interdependency of actions in response to multiple events of different types, can be far more complex than action activation rules governing normal operations. Just

like error handling code, error and failure handling rules must be well designed.

Meeting dependability requirements from the system perspective is even more challenging. For many reasons, including error and failure handling, we may want iGaDs to be able to communicate and collaborate. This and other seemingly reasonable functionalities can add considerable complexity and make the system as a whole less dependable. Problems on tradeoffs amongst functionalities, system complexity and dependability need to be rigorously formulated and solved.

Lastly, but most importantly, we will need to reexamine the message format standards (e.g., CAP [9]) and delivery services (e.g., IPAWS-OPEN [8]). Thus far, the primary consumers of CAP alert messages are public emergency alert systems and commercial mobile alert services. These systems process the messages and then rebroadcast to public in human-readable forms. The open platform, protocols and standards designed for this much more restricted usage may not be ideal when messages are also pushed to enterprise and personal computing and communication platforms running diverse iGaDs used for diverse purposes in diverse environments and conditions. Extensive testing and field trials need to be done to determine whether and what revisions will be needed.

ACKNOWLEDGMENT

The authors wish to thank T. Y. Chen of National Tsing-Hua University for his comments and suggestions.

REFERENCES

[1] C. Buratti, A. Conti, D. Darkari, and B. Verdone, "An overview on wireless sensor networks technology," *Sensors*, pp. 6869-6896, 2009

[2] R. Sherwood and S. Chien, "Sensor Web: a new paradigm for operations," in *Proceedings of International Symposium on Reducing the Cost of Spacecraft Ground Systems and Operations*, June 2007

[3] OSIRIS (Open architecture for Smart and Interoperable networks in Risk management based on In-situ Sensors), http://www.osiris-fp6.eu/

[4] SANY - an open service architecture for sensor networks, edited by M. Klopfer and I. Simons, http://sany-ip.eu/publications/3317, 2009.

[5] J. Strand, "SensorNet," presentation at http://www.ittc.ku.edu/workshops/sensornet/john_strand.pdf

[6] M. Botts, G. Percivall, C. Reed and J. Davidson, "OGC® sensor web enablement: Overview and high level architecture," http://www.opengeospatial.org/pressroom/papers

[7] OASIS (Organization for the Advancement of Structured Information Standards), http://www.oasis-open.org/

[8] IPAWS-OPEN (Integrated Public Alert and Warning System – Open Platform for Emergency Networks), at http://www.fema.gov/emergency/ipaws/about.shtm

[9] CAP: Common Alerting Protocol, V1.2, http://docs.oasis-open.org/emergency/cap/v1.2/CAP-v1.2-os.html ,

[10] EDXL-DE: Emergency Data Exchange Language Distribution Element, V1.0, at http://www.oasis-open.org/committees/download.php/17227/EDXL-DE_Spec_v1.0.html

[11] P. Patel, "Phase-array radar could improve tornado warning time," IEEE *Spectrum,* News, June 2011.

[12] WDSS-II (Warning Decision Support System – Integrated Information), http://www.wdssii.org/

[13] T. C. Shin, *et al*., "Strong motion instrumentation programs in Taiwan," in Handbook of Earthquake and Engineering Seismology, W. H. K. Lee, H. Kanamori and P. C. Jennings, Ed., pp. 1057-1066, Academic Press, 2003 and BATS (Broadband Array in Taiwan for Seismology), http://bats.earth.sinica.edu.tw

[14] FMNEAR Solution (developed by Dr. Bertrand Delouis, 2010)

[15] G. P. Hayes, L. Rivera, and H. Kanamori, "Source inversion of the W-Phase: real-time implementation and extension to low magnitudes," *Bulletin of Seismological Society of America*, 80, pp. 817-822, 2009.

[16] T. Nagao, A. Takeuchi, and K. Nakamura, "A new algorithm for the detection of seismic quiescence: introduction of the RTM algorithm, a modified RTL algorithm," *Earth, Planets and Space*, Vol. 63, No. 3, pp. 315 – 324, 2011

[17] J. W. S. Liu, E. T.-H. Chu and C. S. Shih, "Cyber-physical elements of disaster prepared smart environments," in presss, IEEE *Computer,* http://doi.ieeecomputersociety.org/10.1109/MC.2012.149, April 2012.

[18] D. Cook and B. Das, *Smart Environments: Technology, Protocols and Applications,* Wiley, 2004.

[19] CAP-EAS Implementation Guideline, CAP-EAS Industry Group, 2010.

[20] Common Alerting Protocol Library, http://code.google.com/p/cap-library/

[21] NSXML Parser, at https://developer.apple.com/library/mac/documentation/cocoa/conceptual/nsxml_concepts/NSXML_Concepts.pdf

[22] libxml2, at http://xmlsoft.org/

[23] SAXParser, at http://docs.oracle.com/javase/1.4.2/docs/api/javax/xml/parsers/SAXParser.html

[24] LINQ to XML at http://msdn.microsoft.com/en-us/library/system.xml.linq.aspx

[25] E. C. Chee, E. Mohd-Yasin, A. K. Mustaph, "RBStrex: Hardware XML parser for embedded systems," International Conference for Internet Technology and Secure Transaction, 2009.

[26] CLIPS: A Tool for Building Expert Systems at http://clipsrules.sourceforge.net/

[27] JESS, the Rule Engine for the Java Platform , at http://herzberg.ca.sandia.gov/

[28] E. D. Schmidt, "Logician: A table-based rules engine suite in C++/.NET/Javascript using XML," at http://www.codeproject.com/Articles/194167/Logician-A-Table-based-Rules-Engine-Suite-In-C-NET

[29] C. Choi, et al., "MiRE: a minimal rule engine for context-aware mobile devices," *Proceedings of the 3rd International Conference on Digital Information Management*, November 2008