Sixth Annual IEEE International Conference on Pervasive Computing and Communications

An Autonomic Context Management System for Pervasive Computing

Peizhao Hu [1] [2], Jadwiga Indulska [1] [2], Ricky Robinson [2]
[1] *School of Information Technology and Electrical Engineering*
The University of Queensland, Brisbane, QLD 4072, Australia
{*peizhao, jaga*}*@itee.uq.edu.au*
[2] *NICTA*
300 Adelaide Street, Brisbane, QLD 4000, Australia
Ricky.Robinson@nicta.com.au

***Abstract*—Context-aware applications adapt to changing computing environments or changing user circumstances/tasks. Context information that supports such adaptations is provided by the underlying infrastructure, which gathers, pre-processes and provisions context information from a variety of context information sources. Such an infrastructure is prone to failures and disconnections that negatively impact on the ability of context-aware applications to adapt (and therefore dramatically impact on their usability). This paper describes a model-based autonomic context management system (ACoMS) that can dynamically configure and reconfigure its context information gathering and pre-processing functionality in order to provide fault tolerant provisioning of context information. The approach uses standards based descriptions of context information sources to increase openness, interoperability and scalability of context-aware systems.**

I. Introduction

Context-aware applications use context information to adapt their behaviour and adjust to changing computing environments or changing user circumstances. The provisioning of context information needs to be reliable - the context information must be available when it is needed by applications. It is a challenging requirement as the sources of context information (e.g., sensors) can fail or may become disconnected due to communication failures or mobility of the context-aware application. A common way to provide this reliability is through redundancy of context sources. Having redundant and always active sources of context information [2] it is possible to withstand failures of some context sources; it is also possible to support mobile applications as their context queries/subscriptions can be served by the active context information sources in the environments the applications enter. This approach, however, can be very wasteful in terms of resource usage and it will not scale to large numbers of sensors.

To provide efficient management of resources it is necessary to introduce autonomy in systems for gathering, pre-processing, managing and provisioning context information, which are usually termed *context management systems*. One of the goals of autonomic computing is self-management (self-configuration, re-configuration, self-healing, self-protection and self-optimisation) [10]. Introducing elements of autonomy into the context management system will allow: (i) dynamic and automatic configuration/activation of context information sources when they are needed, (ii) better monitoring of the system to support its fault tolerance, (iii) dynamic replacement of failed or disconnected context sources by sources which can produce the required type of context information (e.g., location information can be obtained by face recognition systems using conventional surveillance cameras in lieu of physical location sensing technologies such as GPS), and (iv) opportunistic use of sensors (discovery of mobile sensors, and location-based and preference-based dynamic binding of sensors).

Context-aware applications have traditionally been developed using one of the following three approaches:

1) each application directly communicates with sensors and other sources of context information, pre-processes the raw data to the required level and evaluates the information to make decisions about how to adapt (*no application-level context model*);
2) applications are developed with the aid of reusable libraries/toolkits for processing context information which assist with gathering and pre-processing data (*implicit context model*); or
3) applications have their own well-defined context models and use a shared context management

0-7695-3113-X/08 $25.00 © 2008 IEEE
DOI 10.1109/PERCOM.2008.56

IEEE computer society

infrastructure to populate the models at run-time using context sources (*explicit context model*).

In the first two approaches it is the responsibility of the context-aware applications to handle faults in context information gathering and pre-processing, which increases the size and complexity of the applications and the difficulty of implementation. The third approach is based on explicit formal models of context information (types of context information and the quality of this information) used by particular applications and therefore allows multiple context-aware applications to share a set of common context sources and context information pre-processing components, limiting the burden on resource constrained context sources. In addition, owing to the models, it allows the issues of fault tolerance, re-configuration and self-healing to be pushed to the context management system (i.e., to the middleware layer), freeing the application developers from this concern and allowing them to concentrate on the desired functionality and business logic of the application.

In this paper we describe the design, prototyping and evaluation of a context model-based, scalable, and autonomic context management system (ACoMS). The system supports dynamic configuration and re-configuration of the set of context information sources required by applications, and therefore achieves fault tolerance without relying on redundancy. It also supports discovery and opportunistic use of sensors. The run-time replacement of context information sources is guided by the requirements that the applications should be provided with the required quality of context information (QoI) and that sources should meet any environmental constraints (e.g., sensor communication should be kept at such a volume so as not to degrade the level of QoS provided to applications executing in the environment).

The structure of the paper is as follows. Section II describes the current approaches to developing reconfigurable context management systems. Section III illustrates an application that requires both dynamic replacement of context information sources and their opportunistic discovery and use. Section IV, describes features of context information models that are needed to support autonomic context management systems and shows a context information model for the application described by the emergency scenario in section III. Section V explains how openness and interoperability can be increased by applying sensor description standards in pre-processing of context information. Sections VI and VII show the architecture and functionality of ACoMS, respectively. This is followed by its evaluation in section VIII and conclusions in section IX.

II. Related Work

Current approaches to gathering and pre-processing context information through the use of shared libraries or toolkits are predominantly based on some kind of component composition: programmatic composition (the programmer builds the composition), specified composition (the composition is described outside the program logic by means of a specification) or dynamic composition (composition occurs automatically by matching interfaces and their semantics). The latter provides some support for reconfigurability as the pre-processing elements can be re-composed in the case of component failure or disconnection. The RUNES middleware [3] is a typical example of such an approach - it enables components to be dynamically deployed and linked, whereby dependencies between component types are expressed in advance, and these dependencies are then satisfied at run-time. This is similar to a variety of earlier component-based approaches like [1], [2], [11].

In [9], the authors use the concept of dynamic service composition to maximise the number of services available for pre-processing sensed data. In this approach, services contributed by a device can be discovered and exploited by neighbouring devices to create composed services for pre-processing of contexts. The dynamic reconfiguration is in this approach supported by device classification, graphs of available services, parameters of services and user tasks. This approach targets a small, closed system.

The main contribution of our approach is that the functions of monitoring and reconfiguring the context information gathering and pre-processing tasks is provided entirely by the middleware (by the context management system), not by the application. Furthermore, it is a context model-based approach in which context models inform the selection of suitable context sources and the dynamic composition of pre-processing elements. Crucially, this means our solution retains the considerable benefits of current model-based approaches including support for high-level reasoning and sophisticated programming techniques. Moreover, to increase openness, interoperability and scalability, in particular to facilitate opportunistic discovery and use of sensors, we use upcoming sensor description standards in our solution.

III. Firefighting Scenario

In this section, we describe an example scenario of firefighting and rescue operations to illustrate context-

aware applications that require both dynamic replacement of context information sources and opportunistic discovery of context sources.

A team of fire-fighters is alerted to an explosion and subsequent fire by the safety monitoring system (SMS) of an office building. The SMS generates this notification by using sensor readings from the sensor network installed in the building. The fire-fighters - each equipped with a heads-up display and a set of tiny body-mounted sensors for detecting smoke, reporting the temperature, sensing movement and so forth - arrive on the scene in a similarly equipped vehicle. One of the vehicles also hosts the command centre system (CCS) - an application that continuously evaluates the situation at the site of an emergency and aids in the preparation of fire-fighting and rescue strategies, taking floor plans and potential risks into account. The CCS also negotiates access to the sensors installed in the building.

The CCS updates multiple displays mounted on the emergency vehicle with information about the location, missions, activities, resource allocations and communication channels of each rescue crew member. This is the primary means by which the incident controller (the officer in charge of the overall operation) maintains awareness of what is happening inside the burning building. Surveillance cameras inside the building that are not affected by smoke or fire use the SMS to deliver to the CCS images or streamed video so that people remaining inside the building can be located, recognised and identified (i.e., to distinguish between victims and their rescuers). Before the human rescue crew members enter the building, a number of sensor-carrying robots are dispatched. The data collected by the robots can be used to forewarn the crew of potential backdrafts in particular areas of the building. The robots can also enter areas that are too dangerous for humans. When the rescue crew members enter the building, their location sensing technology dynamically changes from outdoor (e.g., GPS) to indoor (e.g., The Cricket Indoor Location System[1]) technologies - the change is triggered by the increased quality of location information detected from the indoor sensors. This location information is routed back to the CCS and disseminated amongst the crew members by way of an ad hoc network that is formed between the devices carried by the fire-fighters and robots. As the fire gets worse and sensors installed in the building are damaged, the sensors in the equipment of rescue crew members - which remain in standby mode until required in order to reduce radio interference, network congestion, and battery power consumption - are activated by the system in order to continuously supply information about the current situation.

In the remaining sections we will explain how ACoMS provides reliable context information provisioning for such applications.

IV. Context Models for ACoMS

Context models capture the relevant concepts and relations required by context-aware applications - they abstract raw data gathered from the sources of context information (e.g., sensors). Such raw data may need to be pre-processed to acquire the form defined by the context model. If run-time replacement of a context information source is needed or the sources of context information are to be dynamically configured/activated when the applications start then a mapping is required from the context model to the appropriate source of raw context data. This raw data, when somehow pre-processed, should provide higher level context information with QoI required by the applications. A variety of context models have been proposed for context-aware systems [14]. The most appropriate models for autonomic context management systems are those that define not only the types of context information required by the application, but also its metadata (for example, classification of information types into sensed, non-sensed, and derived types, quality of the required information) that can be exploited to define dynamic, context-aware mappings between context model and their appropriate sources of information. Our own *fact-based* context modelling approach [5], [6] meets this requirement, and is therefore used as the basis for the work presented in this paper. However, it should be noted that our solution for autonomic context management can be used with any fact-based model.

To provide the required mapping, not only the context information model is needed but also some kind of sensor descriptions. We advocate the use of sensor description standards as they support opportunistic discovery and use of sensors and also because the application of standards increases interoperability. However, the sensor description standards describe static features of sensors while sensors are also described by dynamically changing information (e.g., location of mobile or static sensors). We capture this dynamic information by a sensor context model. Figures 1 and 2 illustrate the application context model of the command centre application from

[1]http://cricket.csail.mit.edu/

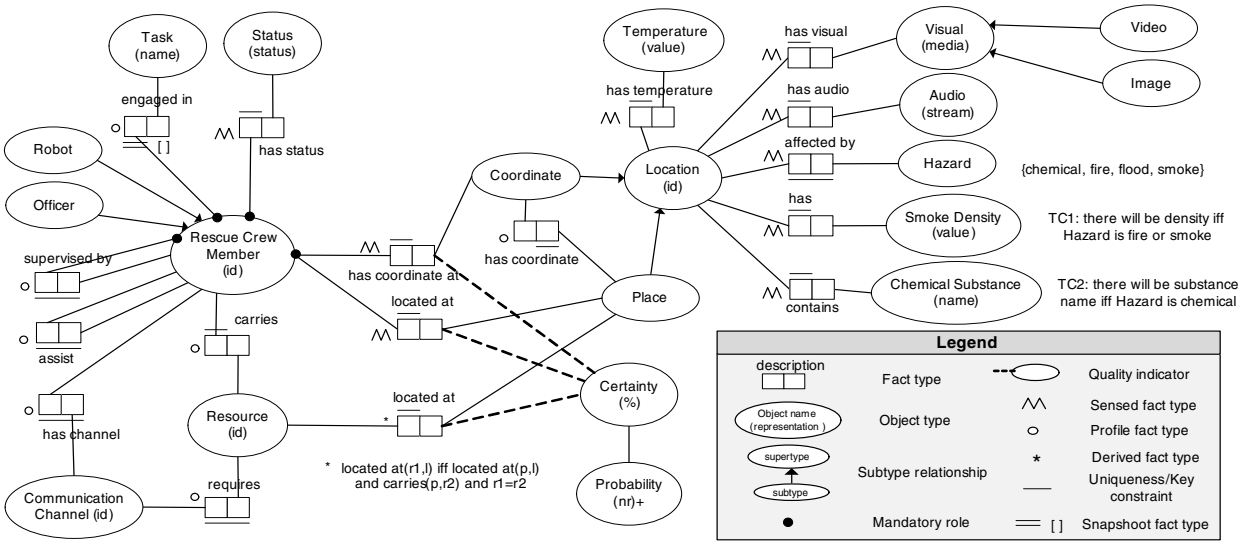

Fig. 1. Command Centre Application Context Model *(Simplified)*

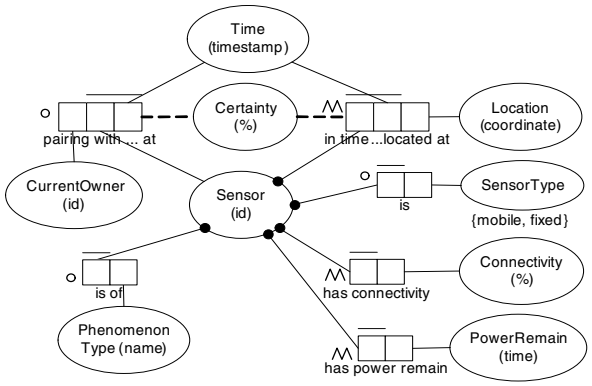

Fig. 2. Sensor Context Model *(Simplified)*

the emergency scenario in Section III, and the sensor context model, respectively.

Three types of context fact are presented in these context models, they are *profiled*, *sensed*, and *derived* fact types. Profiled information is user-supplied, and is therefore initially very reliable, but often becomes out of date, while sensed context information is usually highly dynamic and prone to noise and sensing errors. Derived information is inferred from other contexts using a derivation rule. This classification of information types allows context information to be managed and processed according to the characteristics of its type. Another important property of the context modelling approach we exploited is its ability to capture quality of information, which can be used to support resolution of ambiguous context information. In our example application context model, *certainty* metadata of a context fact type (e.g., rescue crew member *located at* place) indicates the confidence of individual fact instances gathered from sensors.

The application context model captures detailed context information required to model the command centre application. A range of context information is described in this example model, including: (i) the relationship between entities (people, places), and (ii) the properties and activities of entities.

The sensor context model example describes only a subset of the dynamic context information required to model sensors. This context information includes ownership, location, communication connectivity and remaining power. Information presented in Figure 2 can be gathered directly from sensors at the time when they are discovered and registered to the system.

Although many context-aware applications interact with higher level abstractions rather than use context facts directly, it should be noted that these abstractions are built from context facts; therefore when considering fault tolerant context management systems it is sufficient to limit the problem to fault tolerant provisioning of context facts. Therefore we do not discuss higher level abstractions in this paper.

V. Pre-Processing of Context Information

In this section, we discuss our approach to dynamic, context-aware and standards-based composition of data

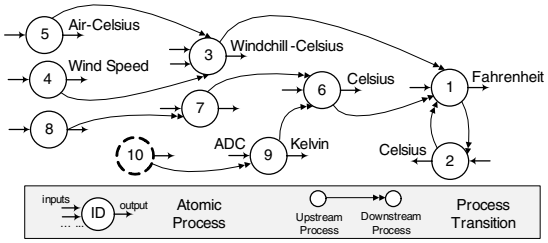

Fig. 3. Example Working Knowledge for Pre-Processing Context; *atomic process 10 is a sensor.*

processes that can pre-process raw context information data into the abstract context facts required by model-based approaches. This composition is required for both self-configuration of context information sources and their run-time re-configuration.

As the heterogeneity of context information sources is one of the obstacles in developing open and interoperable systems for gathering, pre-processing and managing context information, we advocate the use of sensor description standards. SensorML[2] is a sensor description framework, which details information about sensors and models for processing sensor observations. We adopted the SensorML framework to describe *post-discovery* sensor specifications (including identity, classification, characteristics and capabilities) and to apply its data processing model to the pre-processing of context information. SensorML can describe individual processing elements (including sensors and process models), called *atomic processes* as well as compositions of atomic processes, called *process chains*. Each atomic process defines its inputs, its outputs and the methods that yield the outputs from the inputs. These processes can be composed to create process chains, which can be further composed in a *filter-and-pipe* [13] fashion.

In ACoMS, models of generic data pre-processing are described in a system-wide Process Knowledge Base (PKB). As atomic processes are added to the system, they are linked into the existing set of atomic processes by matching their inputs to the outputs of existing processes, and their outputs to the inputs of existing processes. At run-time, ACoMS extracts a subset of the PKB graph such that this subset satisfies the requirements of the application context models. The subset is further constrained to the process chains that provide the best QoI. This subset of the PKB is called the *working knowledge*. Sensors discovered by the context manager will also be modelled as atomic processes in the working knowledge. Figure 3 shows an example graph of the working knowledge, which includes one potential process chain that translates an Analog-to-Digital Converter reading to Kelvin and then to Fahrenheit ($10 \rightarrow 9 \rightarrow 6 \rightarrow 1$) where *atomic process 10* is in fact a sensor discovered by ACoMS at run-time. This graph encapsulates the set of all possible process chains, and it serves as the main data structure on which our composition algorithm operates. The purpose of this algorithm is to find processes or process chains that can deliver information that corresponds to the requirements of the application context model and that satisfies any application specific constraints. In other words, this algorithm matches context fact types to sources of context information by taking into consideration dynamic constraints. Our algorithm, shown below, consists of two passes. The first pass is a *top-down* traversal of the graph, starting at processing elements whose output corresponds with that of the context fact type for which a source is needed, and ending when the traversal reaches a process for which either its inputs and constraints can be satisified by the application request as parameters or no input is required. The constraints may restrict the set of sensors under consideration to particular locations or to those which can meet Quality of Information (QoI) requirements, for example. In short, this top-down pass extracts a feasible process chain from the graph of all processes. At this point, the *bottom-up* phase commences, whereby concrete implementations for each of the processes traversed during the top-down phase are located. If a suitable concrete implementation for a process cannot be found, that process, and any that follow it in the chain, are removed and the procedure continues. Pseudo-code for this algorithm is shown below.

```
FindSource(FactType, Constraints,
  ProcessGraph, ProcessImpls) {
  // Map processes to the FactType
  foreach(Process p in ProcessGraph) {
    if(p.outputType in FactType.roleTypes) {
      FactType.mapTo(p); }}

  ProcessChain chain = {};
  foreach(Process p in FactType.links) {
    chain.add(p);
    ComposeChain(chain, Constraints);
  }
  return chain; }

ComposeChain(Chain, Constraints) {
  Process p = Chain.tail;
  // p is either a ProcessModel or a Sensor
  if(p.isSatisified(Constraints, Inputs)
     || p.numInputs == 0) {
```

[2]http://vast.nsstc.uah.edu/SensorML/

```
  foreach(Process r in Chain) {
    ProcessImpl pi = ProcessImpls.
         FindImplementationOf(r);
    if(pi == null)
      RemoveTailFrom(Chain, r);
    else
      r.AssignImpl(pi); }
} else {
  foreach(Process q in p.inputLinks) {
    Chain.add(q)
    ComposeChain(Chain, Constraints);
  }
}}
```

We've presented a simplified version of our algorithm due to space constraints; it omits the ability to return multiple alternative process chains, disregards cycles in the graph and does not take the length of process chains into account.

VI. The Context-aware System Architecture

The goals of ACoMS are to support openness, interoperability, and scalability of context-aware systems, and to provide fault tolerant provisioning of context information to context-aware applications. ACoMS forms the middleware layer between the context-aware applications and the sources of context information as illustrated in Figure 4. In this section we discuss the models of context-aware applications and context sources required by ACoMS and describe the ACoMS architecture and the role of the ACoMS components. The following section will show how these components provide the autonomic behaviour of ACoMS.

ACoMS is an extension of our previous work. In the past, we have developed a context modelling notation and a context management system able to provision context information for applications and their context models [5], [6]. However, this context management system was not autonomic (i.e., could not self-configure context sources and could not, at run time, replace failed/disconnected context sources). The system had some support for failures/disconnections provided that there was redundancy of context sources. The problem of high network load due to "always on" redundant sensors was somewhat limited by using an event notification protocol [12] that supported quenching (suppression of notifications from the event publishers in the absence of matching subscriptions) between the context management system and the context sources. However, there was no support for enabling/disabling *particular* redundant sources, so that all sources satisfying a context fact type in an application context model would produce data when only one source was really necessary. Furthermore, specialised wrappers needed to be developed for each context source, and no use was made of standards, limiting the ease with which new types of context sources could be incorporated. These challenges have also been unmet by other context management systems. In Figure 4 shaded boxes indicate new components added to our previous context management system to achieve the aforementioned reconfiguration ability.

A. Context-aware Applications

Context-aware applications are consumers of context information and base their adaptation decisions on that information. The goal of ACoMS is to provide fault tolerant context provisioning for multiple context-aware applications. Our previous context management system used models of context information fact types required by each application (shown as Application Context Model in Figure 4), a preference model for deciding amongst a set of possible choices with reference to the current context (in the form of higher level constructs called *situations*), and a triggering model enabling actions to be invoked in response to changes in the current situation. Each of these concepts is carried over to ACoMS. The Application Context Models are used to govern the population of the database of context facts (shown as Application Context Facts) that supports all applications with the context information facts gathered from context sources. One more type of information related to the applications is required in order to support ACoMS: Application Context Subscriptions that indicate the application specific requirements for context provisioning (e.g., QoS and QoI of the required context information).

Context information facts managed by the context management system may be shared by the applications; however, ACoMS has to deliver this information to each application with the QoS and QoI requested by each particular application.

B. Context Sources

Diverse context sources (physical and logical sensors) comprise the context sensing layer in a context-aware system producing sensing phenomenon measurements (i.e., temperature readings, presence status, events of interest, etc.). This information will eventually be converted into context information facts required by applications. It is important that the description of context sources supports their opportunistic discovery. The upcoming IEEE 1451 family of standards[3] for sensor

[3]http://ieee1451.nist.gov

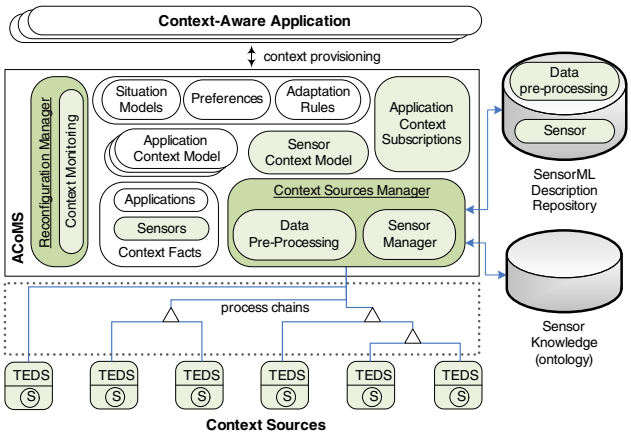

Fig. 4. Open Context-Aware System Architecture

descriptions is being developed to increase sensor interoperability and to allow automatic machine interpretation of the sensor description and therefore meets the opportunistic discovery requirement.

IEEE 1451 provides a way to describe sensor specifications in an on-board memory device, called TEDS (Transducer Electronic Data Sheet), and enables a sensor to introduce itself to the management system with which the sensor can communicate. IEEE 1451.4, a member of the standards family, further refines the TEDS encoding/decoding with templates and introduces virtual TEDS[4], so that a wide range of compact transducers (i.e.: accelerometers, microphones, strain gauges, thermocouples, thermistors, etc.) can also benefit. A TEDS template, written in Template Description Language (TDL), details, bit by bit, the meaning of the TEDS data, and specifies instructions about how TEDS data should be decoded and extracted from its binary encoding. A virtual TEDS offers an alternative file-based specification, accessible from repositories, for sensors which lack storage capability. IEEE 1451 has rapidly become a manufacturer-preferred standard specification for future smart and plug-n-play sensors in various research and industrial domains. This approach allows smart sensors to advertise themselves and self-describe their sensing as well as processing capabilities to higher level management systems and therefore allows adding dynamic and autonomic functionalities to the management systems. ACoMS supports sensors that are described by on-board or virtual TEDS.

[4]http://www.ni.com/teds/

C. ACoMS Architecture

The core part of ACoMS is formed by models, supporting knowledge, and repositories of instances of information described by the models. These elements include: (i) application related models and repositories as described in subsection A, (ii) context models of sensors that capture dynamic sensor information as discussed in Section IV, (iii) a subset of human knowledge for the interpretation of sensor specifications and for the understanding of data processing flows as discussed in Section V, and (iv) a repository of SensorML specifications of context information sources which describe their identity, classification, characteristics and pre-processing capabilities and are used for self-configuration and reconfiguration of mappings between context sources and context facts required by applications. Each SensorML description is dynamically created based on a TEDS description and is described in Section VII. In addition, ACoMS comprises three managers:

Application Context Subscription Manager stores the subscriptions that are defined by application designers in order to capture QoI and/or QoS requirements (including the type, frequency and granularity of information applications need along with other additional constraints). This information is then served as application specific knowledge for context provisioning and for monitoring of the context provisioning.

Context Source Manager manages low-level communications with/for context sources, provides seamless context sources discovery, registration and configuration services. It also pre-processes raw context data to high-level context information facts through a chain of dynamically assigned atomic processes. Figure 4 shows a logical view of the example process chains that are dynamically formed using diverse context sources.

Reconfiguration Manager performs cross-layer context monitoring throughout the system and reconfigures/reorganises, in co-ordination with the Context Source Manager, mappings between context information facts and their appropriate sources.

Figure 5 illustrates the mappings as context information flows from context sources through the dynamically assigned process chain to context fact types requested by applications. The data is processed by the distributed implementation of atomic processes in a process chain, denoted *process executions*.

VII. Degree of Autonomy in ACoMS

In this section we discuss in more detail the self-configuration, reconfiguration and self-healing capabil-

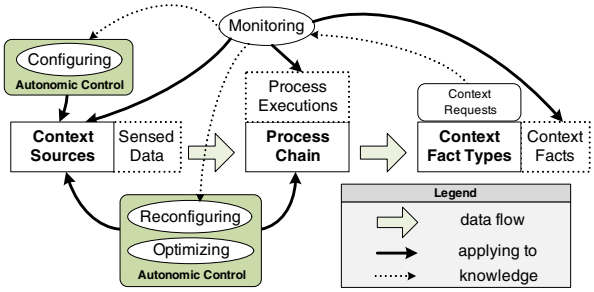

Fig. 5. The Autonomic System Architecture of ACoMS

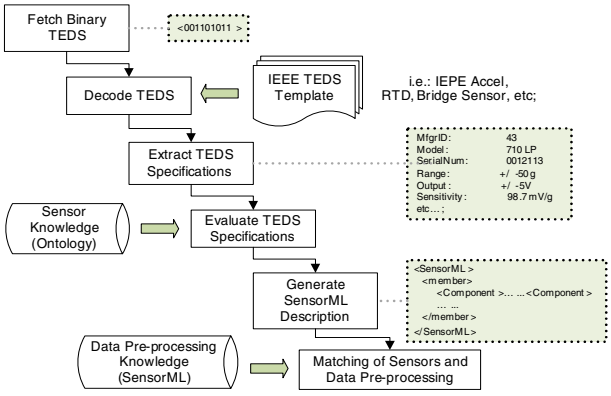

Fig. 6. Seamless Discovery and Self-Configuration of Context Sources; snapshot example data in each step is presented in dotted grey boxes.

ities of ACoMS. The self-optimising and self-protecting features of autonomic systems are not fully explored yet and will be addressed in our future work. Figure 5 shows these features of ACoMS in an autonomic system architecture.

A. Seamless Self-Configuration of Context Sources

In [7], [8], we have already discussed how IEEE 1451 and the SensorML description framework provide a way to describe sensor specifications and data processing. However, a TEDS template describes very limited sensor properties and capabilities (i.e., hardware characteristics, sensor calibration, sensing phenomenon), whereas SensorML can capture detailed information, but the description cannot be processed by resource constrained devices. A hybrid approach is, therefore, proposed, which uses TEDS for enabling sensors to describe their low level specifications, and then SensorML to model these descriptions in an extensible and flexible representation for high-level systems/applications. Figure 6 shows this hybrid approach of dynamically mapping TEDS specifications into corresponding SensorML descriptions.

At the time when sensors are discovered through their physical communication interfaces, they self-introduce/advertise their TEDS, and then, as shown in the diagram, the retrieved TEDS binary data is decoded using the assigned TEDS templates. Sensor specifications knowledge, in the form of an ontology knowledge base, is exploited to provide semantic mappings of each TEDS specification to its corresponding SensorML description. And as a result of this procedure, a SensorML description of sensor specifications is generated and stored in the SensorML repository. Accordingly, the data processing knowledge is applied to compute appropriate data processing solutions for the newly registered sensors. This *sensor* and matching *process* information is then captured, as part of the dynamic context information of sensors, and used to populate the sensor context model, as illustrated in Figure 2. This dynamic sensor information will be used in the evaluation of appropriate data processing solutions for the requested context fact types, as discussed in Section V. When the sensor is registered, the middleware system then checks to see whether there is any matching context request from applications, otherwise these sensors will be advised to go into sleep mode.

B. Provisioning of Application Context

For application requests, we adopt both querying and the publish/subscribe paradigm. Applications specify their context provisioning requirements, including the type (*query* or *notification*), frequency (*one-time* or *periodic*) and granularity (*accuracy*) of information along with other relevant constraints in the form of context request subscriptions.

ACoMS can merge/combine different application context requests which have common QoI factors [4]. For example, temperature information may be requested by three different applications (as requests *ReqA*, *ReqB* and *ReqC*) with slightly different requirements, but can be combined dynamically in the following way for sharing of raw sensed data from the same context source.

```
ReqA={mode=query,accu=1};
ReqB={mode=pubsub,freq=10s,accu=5};
ReqC={mode=pubsub,freq=300s,accu=10};
======> ReqApps={freq=10s,accu=1};
```

In this example, ACoMS adjusts the underlying sensing infrastructure to produce sensed data at the specified sampling rate and granularity for both applications (*ReqApps*) according to the common factors of *frequency* and *accuracy*.

C. Monitoring

The goal of the monitoring protocols in ACoMS is to monitor whether the context source-context fact mappings are alive (i.e., provide the required context information with the required frequency and QoI). As discussed earlier, the classification of context fact types (*profiled, static, sensed and derived*) together with application context subscriptions provide information about how context information should be gathered, while the SensorML descriptions and the context models of sensors describe both static and dynamic sensor information. Using this knowledge, monitoring protocols of different types can be designed to support ACoMS. The full design and development of such monitoring protocols is left as future work. Currently reconfiguration is triggered in ACoMS by very simple sensor monitoring protocols. Figure 5 illustrates monitoring required by ACoMS.

D. Reconfiguration of the context source - fact mapping

When a context-aware application is executed, ACoMS knows the mapping of context sources to context facts required for this application. This mapping can fail (i.e., context facts are not delivered or the required QoI is not provided) for two reasons as illustrated in Figure 5: failure of one individual element (context source or process execution) or failure of the entire process chain. Therefore we describe two types of reconfiguration: (i) finding and replacing individual elements in a context mapping; and (ii) rebinding the mapping for the requested context fact type to a new process chain and new matching context sources.

The monitoring service of the Reconfiguration Manager (RM) provides feedback information about the operational status of individual elements. When failures of these elements occur, the RM notifies the Context Source Manager (CSM) and provides it with information about the failure of the specific element. The CSM then carries out a reconfiguration procedure to replace the failed elements if alternatives exist. To reconfigure from context source failures, the CSM looks up the currently available sensors from the repository of sensor context facts for the matching replacement, whereas reconfiguring the defective process executions requires the CSM to either discover alternative execution instances of the process model or re-initiate a new execution instance according to the SensorML process chain descriptions. In the case where there are no alternatives, the CSM will try to find a new process chain to replace the defective one as described in Section V.

VIII. Evaluation Study

For the purpose of evaluating the ACoMS prototype, we conducted several evaluation studies, including: (i) evaluation of our standards-based approach to dynamic discovery and configuration of context sources; (ii) evaluation of dynamic composition of data processing knowledge; and (iii) a scenario-based evaluation of the command centre system applications.

Firstly, we simulated the *smart* sensors by incorporating the aforementioned sensor description standards (i.e., IEEE 1451 and SensorML) in our experimental set-up consisting of 10 Tmote Sky[5] sensor nodes and four networked cameras with different specifications. The purpose of this evaluation was to verify the approach of adopting sensor description standards in supporting self-configuration of our system. Therefore, we described the sensor specifications accordingly in the form of TEDS, and stored this information either in the on-board memory for the Tmote sensor nodes or in a repository for networked camera sensors (the cameras can retrieve it with their assigned model ID).

Each Tmote sensor runs a TinyOS[6] application we developed for basic management tasks (e.g., registration, sleep, sampling and wait-for-service).

In the experiment, Tmote sensors advertised their presence to the environment. When the system detected the sensor, it went into a registration mode and asked the sensors to provide their TEDS binary specifications (or model information for retrieving its SensorML description from the repository). The system then followed a registration procedure, as described in Section VII.

To prove the practicability of our dynamic composition of data processing knowledge on various system configurations, we developed a prototype knowledge base system, which allows us to add data processing models and dynamically compose them together as an interconnected knowledge graph as we described in Section V. We conducted several experimental stress-tests (adding, removing of large number of process models and searching of process chains) on various system configurations (from single-board router devices, RB532A[7], to standard single-core desktop) and knowledge base sizes. Figure 7 shows the experiment results (each averaged from 10 individual tests) in finding a matching process chain for a given context fact type. These results show that for the most portable mobile

[5]http://www.moteiv.com
[6]http://www.tinyos.net
[7]http://www.routerboard.com

KB Size (Processes)	Platforms, Time (Millisecond)		
	266MHz 64MB	1GHz 512MB	3GHz 1GB
100	15.159	0.151	0.044
1000	148.689	1.502	0.194
10000	1435.546	16.750	5.876

Fig. 7. Feasibility study under various knowledge base sizes and platform configurations

devices (with MIPS 266MHz processor and 64MB memory) with knowledge base size up to 10000 processes, the time delay for finding a process chain consisting of two processes is averaged to 1.4s, whereas it takes only 16ms for our experimental wireless routers.

Finally, we developed three simple, but fully functional, context-aware applications: a command centre application, as shown in Figure 8, and two lightweight rescue crew applications (different in the ways they communicate with the ACoMS and the requirements of context information for their specific operations) based on the scenario described in Section III. These applications were developed and exploited to evaluate fault-tolerant as well as scalability objectives discussed in Section VI. The command centre application shows various, up to date, context information (including temperature, visuals and audio) of the target locations in the view of a floor plan for operational planning, while the two other lightweight applications show a selective set of information to rescue personnels (due to the consideration of limited display capability in mobile devices). At runtime, applications request context information from the ACoMS with their desired QoI and QoS. The ACoMS automatically configures sources of information, which can be physical sensors or virtual/logical sensors (such as web services). If multiple application context models contain the same fact type, only a single binding from the fact type to a context source is required, lowering the burden on context sources (as discussed in Section VII). In our previous system, support for overlapping requests from applications was less sophisticated as it did not take into account the differing QoS and QoI requirements of the applications. We conducted the following experiments: (i) redundant sensors were started, and the system compared the accuracy of the new sensors with the existing sensors and triggered a replacement when appropriate; disqualified sensors were advised to go into wait for service mode; (ii) then, we simulated sensor failures by turning sensors off, and the system verified the existing sensors against the information requirements of the application requests and advised the sensors to start sampling at the required sensing frequency in order to replace the failed sensor.

IX. Conclusions

In this paper, we presented the design, prototype and evaluation study of an autonomic context management system that provides both self-configuration of context sources required by context-aware applications and run time replacement of failed/disconnected context sources. The configuration/reconfiguration is carried out by the context management system (i.e. at the middleware layer) providing applications with fault tolerant provisioning of context information. The replacement of context sources relies on some redundancy of context sources; however, a sensor does not need to be replaced by the same type of sensor and disqualified sensors, as well as those surplus to requirements, can be suppressed. The new sensor can be of a different type provided that its sensed data can be pre-processed to the required context fact.

To achieve such autonomy the proposed context management system is model-based - it relies on the models of context information required by applications, on context models of sensors describing dynamic sensor information, on standards based sensor descriptions detailing static information of sensors and on expert knowledge of pre-processing context information for particular types of sensors. An integration of two sensor description standards, IEEE 1451 and SensorML, facilitates opportunistic discovery of sensors and their self-configuration for delivery of context information. The use of sensor description standards therefore increases openness, interoperability and scalability of context-aware systems. The proposed solution also saves energy, communication and processing resources, as sensors can be attached to the context management system and activated on the fly.

X. Acknowledgements

1) The authors would like to thank Karen Henricksen for her insightful comments on early drafts of this paper.
2) NICTA is funded by the Australian Government's Department of Communications, Information Technology, and the Arts; the Australian Research Council through Backing Australia's Ability and the ICT Research Centre of Excellence programs; and the Queensland Government.

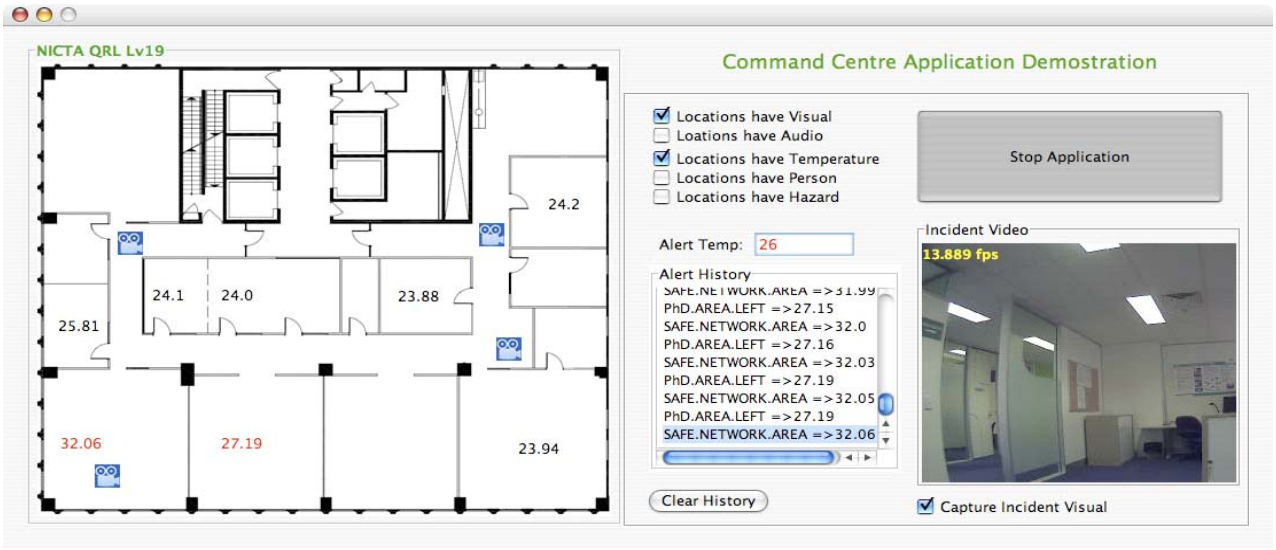

Fig. 8. Command Centre Application Demonstration

REFERENCES

[1] Guanling Chen, Ming Li, and D. Kotz. Design and implementation of a large-scale context fusion network. In *Proc. of the First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services (MOBIQUITOUS2004)*, pages 246–255, Aug. 2004.

[2] Shiva Chetan, Anand Ranganathan, and R. Campbell. Towards fault tolerance pervasive computing. *IEEE Technology and Society Magazine*, 24(1):38–44, Spring 2005.

[3] P. Costa, G. Coulson, R. Gold, M. Lad, C. Mascolo, L Mottola, G. P. Picco, T. Sivaharan, N. Weerasinghe, and S. Zachariadis. The runes middleware for networked embedded systems and its application in a disaster management scenario. In *Proc. of the Fifth Annual IEEE International Conference on Pervasive Computing and Communications (PerCom'07).*, pages 69–78, March 2007.

[4] A. Crespo, O. Buyukkokten, and H. Garcia-Molina. Query merging: Improving query subscription processing in a multicast environment. *IEEE Trans. on Knowledge and Data Engineering*, 15:174–191, 2003.

[5] K. Henricksen and J. Indulska. A software engineering framework for context-aware pervasive computing. In *Proc. of the Second IEEE Annual Conference on Pervasive Computing and Communications (PerCom'04)*, pages 77–86, 2004.

[6] Karen Henricksen and Jadwiga Indulska. Developing context-aware pervasive computing applications: Models and approach. *Journal of Pervasive and Mobile Computing*, 2(1):37–64, Feburary 2006.

[7] Peizhao Hu, Jadwiga Indulska, and Ricky Robinson. Reconfigurable middleware for sensor based applications. In *Proc. of the 3rd International Middleware Doctoral Symposium (MDS'06)*, page 5, Melbourne, Australia, November 2006. ACM Press.

[8] Peizhao Hu, Ricky Robinson, and Jadwiga Indulska. Sensor standards: Overview and experiences. In *Proc. of The 3rd International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP 2007)*, Melbourne, Australia, 3-6 December 2007.

[9] Swaroop Kalasapur, Mohan Kumar, and Behrooz A. Shirazi. Dynamic service composition in pervasive computing. *IEEE Transactions on Parallel and Distributed Systems*, 18(7):907–918, July 2007.

[10] Jeffrey O. Kephart and David M. Chess. The vision of autonomic computing. *Computer*, pages 41–50, 2003.

[11] Hui Lei. Context awareness: a practitioner& perspective. In *Proc. of the International Workshop on Ubiquitous Data Management, (UDM2005)*, pages 43–52, April 2005.

[12] Bill Segall, David Arnold, Julian Boot, Michael Henderson, and Ted Phelps. Content based routing with elvin4. In *Proc. of AUUG2K Conference and Tutorial*, Canberra, June 2000. System Magazine.

[13] M. Shaw and D. Garlan. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, 1996.

[14] T. Strang and C. Linnhoff-Popien. A context modelling survey. In *Proc. of the First International Workshop on Advanced Context Modelling, Reasoning And Management*, Nottingham, England, 2004.