Fault-resilient Ubiquitous Service Composition

Jinchun Xia*, Carl K. Chang*, Tae-Hyung Kim*, Hen-I Yang†, Raja Bose† and Sumi Helal†

*Iowa State University, {jxia, chang, thkim}@cs.iastate.edu
† University of Florida, {hyang, rbose, helal}@cise.ufl.edu

Keywords: *fault-resilient ubiquitous services, ubiquitous service composition, service composition optimization, pervasive computing*

Abstract

Service-oriented architecture (SOA) promises an elegant model that can easily handle dynamic and heterogeneous environments such as pervasive computing systems. However, in reality, frequent failures of resource-poor and low-cost sensors greatly diminish the guarantees on reliability and availability offered and expected by SOA. To provide a framework for building fault-resilient service-oriented pervasive computing system, we present a solution that includes a virtual sensor framework and a high performance service composition solution based on: (1) WS-Pro, a probe-based performance-driven service composition architecture, and (2) an abstract service composition template (ASCT) approach. The concept of virtual sensor enhances the availability of services, while the service composition solution ensures the system can efficiently adapt to changes and failures in the environment. This approach also includes a novel probe-based monitoring technique to actively collect performance data, and the use of an extended Petri net model, FPQSPN, to model performance of service composition.

1 Introduction

Service Oriented Architecture (SOA) has established itself as a "prevailing software engineering practice" in recent years [7]. Its popularity extends to the domain of pervasive computing. Its characteristics of loose-coupling, stateless, and platform-independence make it an ideal candidate for integrating pervasive devices. While the semantics of SOA are being standardized, its use in pervasive computing is under research and being experimented.

With all the promises and favorable characteristics of SOA for dynamic and heterogeneous systems in pervasive computing, it is sometimes easy to forget that underneath all the nice wrappings as highly-reliable and self-integrating services, the real data sources are mass-deployed low-end sensors that are poor in terms of resources available, and inherently unreliable both because of the large number of entities deployed and the often implementation choice of employing low-cost components with no guarantees in their quality.

Unlike web services which are mostly hosted by high-end servers and data centers, with which service failures are rare, ubiquitous services in a pervasive computing system need to embrace service failures as the norm. For these services to work properly and reliably, we need to consider how to improve their availability and how to assess the quality of data they provide so necessary adjustments can be made.

1.1 Fault-resilient Ubiquitous Services

To build a fault-resilient pervasive computing system, we look for measures that can drastically improve the availability and adaptability of services. To enhance the availability, we deploy multiple or even redundant sensors in the vicinity, so small number of failed sensors can be compensated for by readings from their neighbors. To ensure services can adapt to a changing environment, we design a framework that allows fast dynamic re-planning so the failed service can be quickly replaced with substitutes before causing failure of the entire end-to-end service.

The most intuitive way to represent sensors in a SOA is to simply wrap each physical sensor as an individual service implemented in software. However, by bundling sensors with similar functionalities in close proximity, and representing these sensors collectively as a single service, we not only improve its resiliency against failure, but also provide the means to gauge the quality of data collected.

SOA allows reuse and integration of existing software components via service composition. Values are added through the service composition procedure -- the most critical process of service-oriented computing. The nature of service-oriented architecture allows that the service composition to be either static or dynamic. Compared to standard Web Services featuring business logic, service-oriented ubiquitous computing is much more domain-specific and poses additional challenges such as lack of flexibility in re-planning due to context and hardware constraints. Since failure is the norm in pervasive computing, dynamic service composition and re-planning is the only way to ensure uninterrupted services.

Not only do services need to be composed dynamically amid failures and changes, the composition and planning process has to be performed efficiently. Performance has been identified as the most critical attribute of quality by many researchers [6], and Srivastava and Koehler [8] also described the real demand for robust and verifiable web services with high performance from industry. Our experience in the Gator Tech Smart House [13] further confirms that poor performance in service adaptation can seriously hinder the efforts to adopt service-oriented architecture as the solution for pervasive computing.

Our proposed solution for building a fault-resilient pervasive computing system includes two parts. The first part is the concept of Virtual Sensor [14] with the supporting framework to improve the availability of component services. The second part is a probe-based architecture with optimal performance in service composition that can efficiently model, monitor and re-plan the process of service composition. In this architecture, WS-Pro [9][10], the probe-based web service composition architecture is adjusted and supported with the abstract service composition template (ASCT), a template-based service composition scheme to provide a generic solution to high-performance ubiquitous service composition.

1.2 Basic Component Ubiquitous Services

Ubiquitous services can be categorized into three types of abstract service elements based on the functionalities they perform and the roles they play: context providers, context processors and information deliverers. A typical end-to-end service, as depicted in Figure 1, consists of a context provider, a context processor and an information deliverer.

The context provider is the abstract service element that retrieves a context-specific data from hardware components. In other words, a context provider is a wrapper for input sensing components in detecting a specific context. In addition, each context provider has tiny test functions implemented to report the aliveness of hardware components belonging to it. For example, a weather monitor as a wrapper of real or virtual sensors can obtain the current temperature, humidity and windiness outside of the house.

The context processor deals with data from context providers or a database and produces the meaningful context-based information for information deliverers. A context processor may be required to be connected with a particular set of context providers. For example, a personal scheduler searches timetables in a database based on user location, time and weather to provide the customized travel information for a user.

The information deliverer is also a wrapper of a specific hardware component, such as a monitor, a printer, or an emergency alarm, used to present the information generated by the context processor. An information deliverer includes a transcoding feature that transforms the information created by the context processor into a more appropriate format. For example, a printing deliver creates a PDF file based on the information fed from context processors and sends it to a printer.

There are two kinds of paths in the composition template: the paths between context providers/ information deliverers and context processors (in the form of software services), and the paths among multiple software context processor services. The former path is usually performance critical and where failures and changes most often arise. Although context processors can themselves be end-to-end services hierarchically composed with nested context processor and context providers/information deliverers and still be highly related to the hardware devices, this association is considered indirect. Therefore we can still pinpoint the critical path to be the path within the nested service provider between the component context providers/information deliverer and its direct-linked service processor.

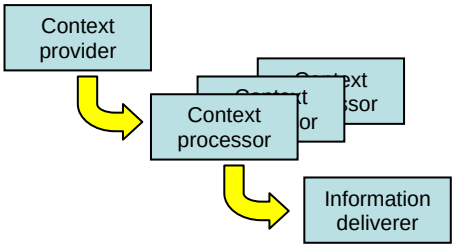


Figure 1: Composition of an end-to-end ubiquitous service. Two kinds of paths are established during composition: (a) paths between context processors and context provider/information deliverers (mandatory), and (b) paths among multiple context processors.

The rest of this paper is organized as follows. We start by introducing the solution to fault-resilient ubiquitous services. In section 2 we present the first part of the solution, the concept and the framework of virtual sensors. In section 3 we discuss about performance management of ubiquitous service composition, and then present the WS-Pro/ASCT approach for efficient service re-planning and composition. A preliminary evaluation on both parts is presented in section 4 and section 5 concludes this paper with summary and future work.

2 Virtual Sensors

We define a virtual sensor as a software sensor service entity consisting of a group of sensors annotated with associated knowledge which enables it to provide services beyond the capabilities of any of its individual component sensors. Virtual sensors may be composed of a group of physical sensors or other virtual sensors.

Virtual sensors can be classified into three categories, *Singleton*, *Basic* and *Derived virtual sensor*, where Singleton virtual sensor represents a single physical sensor, a basic virtual sensor is composed of a group of singleton virtual sensors of the same type, and a derived virtual sensor is composed of a group of basic and/or other derived virtual sensors of heterogeneous types.

Virtual sensors enhance the reliability and availability of sensor services. It provides certain basic guarantees of functionality and is capable of estimating the reliability of data originating from the sensors. It also has mechanisms for recovering from failures of multiple physical sensors and detecting degradation in their performance.

We implemented the virtual sensor framework on top of OSGi, and the architecture of the framework is shown in Figure 2.

The knowledge base manages information related to virtual sensors, such as sensor model definition for each type of virtual sensor, phenomena definition associating each phenomenon with a certain type of virtual sensor, and reliability and availability parameters for virtual

sensors.

The framework controller is responsible for receiving queries from applications and then determining which virtual sensors need to be created, with the help of the knowledge base. Dynamic creation and organization of virtual sensors in the framework allows the sensor network to fulfill queries on phenomena detection despite failures and changes. When a query is issued to detect certain phenomena, the framework controller identifies from the knowledge base, the type of virtual sensor that can detect that phenomenon. It then instantiates the target virtual sensor based on the sensor model definition expressed as ASCT. If the target virtual sensor relies on other virtual sensors as data sources, it tries to subscribe to the data if an instance of those source virtual sensors already exists or instantiates them otherwise.

Within each basic virtual sensor, we constantly monitor the correlated behavior of component singleton virtual sensors, and record whether the differences among their readings are within acceptable margin. This record is used to calculate the weight factor *W* if particular singleton virtual sensors failed and the readings from other singletons were used to approximate their readings. The similarity factor *p* between two sensors is the probability that the difference between their data readings is within a predetermined acceptable margin ϵ. The more similar the readings are between two singleton virtual sensors, the larger the similarity factor *p*, and therefore the larger the value of weight *W*, assigned to one when the other dies. This compensation mechanism mitigates the effect of failed sensors, and enhances the overall availability of the sensor data.

We further define a Data Quality Indicator (DQI) associated with each virtual sensor, which measures the reliability and confidence level of its data. DQI is calculated as follows:

$$DQI = 100 \times \frac{(num\_of\_sensor\_alive + \sum_{s \in all\_failed\_sensor} (1 - e^{\frac{-t}{a}}) W_s)}{total\_num\_of\_sensor}$$

Where *t* is the time elapsed since the sensor network started, and *a* is a constant greater than 0 whose value depends on the sensor and the environment in which it is deployed. DQI values range between 0 and 100.

A basic virtual sensor will have a DQI value of 100 if all its member sensors are functioning properly. The formula adjusts the penalty caused by the discrepancies between the failed sensor and the sensor used to compensate the failure. The term $(1\text{-}e^{-(t/a)})$ is used to factor in the confidence level associated with a particular weight factor W_s based on the length of observation. If a W_s value is calculated using data collected over a longer duration, $(1\text{-}e^{-(t/a)})$ is larger, which means it carries more weight than the ones calculated over shorter observations. To ensure the data quality and service reliability, we define DQ_T as the threshold of reliable data quality. Any sensor readings annotated with DQI value below DQ_T are deemed unreliable, and the source virtual sensor is taken offline to preserve the integrity of the system.

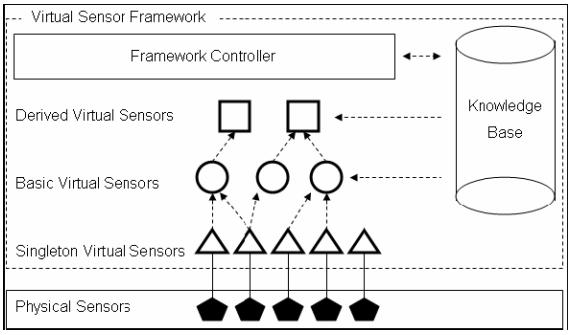


Figure 2: Architecture of Virtual Sensor Framework

3 Efficient Ubiquitous Service Composition

3.1 Performance management of ubiquitous service composition

Service composition has been well studied in web service community, and the wide adaptation of SOA in pervasive computing inspires researchers to examine if the techniques designed for web services are equally applicable to ubiquitous services. There exist subtle but critical differences between the two, however. For instance, in web service composition it is assumed the underlying Internet infrastructure is universal so that services can always be discovered and composed regardless of the differences in platform or communication medium used. The pervasive services, however, are tightly bound to heterogeneous hardware platforms and communication mediums, making their composition different. For example, the discovery of Bluetooth services is limited by the range of the Bluetooth-bound device. This limitation imposes additional challenges in composition of ubiquitous services.

Hummel identifies fault-tolerance as a crucial issue in pervasive services [5][2], and he also pointed out that it is very important that ubiquitous services should be able to "reacting to dynamic changes in time". Our experience in the Gator-Tech Smart House agrees with this assessment. Different smart devices are represented as collaborating services in the service-oriented framework. However, just because they work properly collaboratively in terms of functionality does not guarantees satisfactory services to users should they fail to deliver the service within the timing constraint. In any typical ubiquitous computing environment such as Smart Home, the concern for safety and security are very real and the services addressing these issues have to be delivered promptly. In addition, users usually expect such environment to respond in a reasonably short amount of time. Therefore timely delivery of services is critical. By optimizing the reconfiguration and re-composition of ubiquitous services, we improve the fault-tolerance as well as customer satisfaction.

We observe that context providers and information deliverers encounter more performance problems than context processors. The problem may exist either on the hardware layer (i.e., device connectivity failure) or the service layer (protocols, service failures, etc.). Therefore

there exists a need of monitoring techniques to verify and improve system performance. It is very important to monitor both the link between services bounded by physical device and context processors, as well as the link between the physical devices and their corresponding services, which can be either context provider or information deliverer.

The first link can be monitored by the ubiquitous service composer using probing techniques. The second link can not be checked by the monitoring system because not all hardware components are capable of reporting their current status or need to transmit data regularly. As a result, the probe has to send test messages to locate problematic hardware components. An alternative is to embed a checking mechanism when wrapping the device in a corresponding singleton service so that the health of the devices can be monitored.

Performance management of service composition is an ongoing research in Web Services, and there are two main issues need to be address. The first is the need for an active monitoring technique on service performance. The second is a mechanism to actually improve the performance of service composition procedure itself. In service-oriented ubiquitous computing, the latter one is even more crucial in providing timely reaction to dynamic changes, as ubiquitous services need to adopt their compositions in a timely fashion. Many ubiquitous services also need to provide real-time services to users and other services.

Unlike traditional distributed software performance modeling, the cross-organization and cross-location properties exhibited by ubiquitous services render many existing performance analytic techniques useless. On top of that, during the execution of the composite service, component services can disappear and remerge at any time, delaying verification and testing from design/implementation stages to deployment stage. Run-time monitoring then becomes the best solution in assessing services' properties when traditional verification techniques are insufficient to provide enough assurance [1]. Unfortunately, most existing monitoring techniques do not provide advance reference without historical data. Moreover, because of the dynamic and real-time nature of services, current status is more desirable than historical data. Therefore, passive monitoring is not able to provide enough support for service composition. A more active technique to monitor the runtime performance status of services and provide timely support for dynamic service composition is needed. Zeng et al. suggested the use of probe as one possible solution [12]. In this paper, we employ a testing-based probe for active monitoring.

3.2 WS-Pro

WS-Pro [9][10] was developed to model and optimize system performance in standard web service infrastructure. A testing-based probe was used to actively collect run-time performance status of component services. It supports runtime service composition and composition re-planning. A technique based on Petri net is used to model the stochastic property of the service composition. This technique helps to select the best candidate services to optimize the performance of the composite service and makes automatic verification of the composition possible. A composition re-planning algorithm was designed based on Petri net truncation. The architecture of applying WS-Pro in ubiquitous service composition is illustrated in Figure 3.

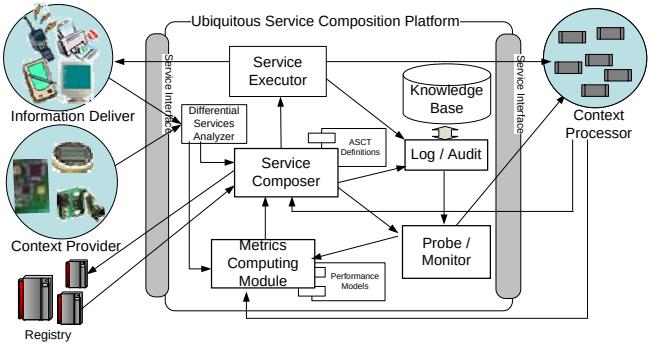


Figure 3: Overview of the ubiquitous service composition

3.3 Abstract service composition template (ASCT)

An abstract service composition template (ASCT) is used to describe the sequential process of critical functions to deliver a composed service. A similar approach using service template is shown in [11]. The ASCT consists of a critical flow of a service using abstract service elements that perform critical functions. Based on the abstract service elements in an ASCT, composite services can be materialized by searching and selecting appropriate actual service instances such as a context provider with assorted virtual sensors. The authors implemented a composition engine, evaluated its performance in terms of processing time and memory usage, and discussed the suitable size of the categories and the number of SEs in each category. The novelty of this approach is the introduction of an abstract layer for describing composite services and the support of their automatic composition. As a result, scenarios of a composite service can be easily described without considering strict and detailed interface description of the intended composite services. Furthermore, this makes it possible for users to create their own services or customize existing ones.

3.4 WS-Pro with ASCT Approach

As described earlier, WS-Pro was originally designed for dynamic web service composition. It does not work well with ubiquitous services because of the following two reasons. First, dynamic composition is important in Web Services because there is always a large pool of candidate component services. With pervasive computing, however, the options are often much more limited. Hence making WS-Pro improper for service composition. Second, each device in pervasive computing, even the ones as small as a sensor are represented as atomic services, resulting in a huge number of nodes need to be modeled. Standard Petri net or GSPN, which is used in WS-Pro, does not scale well; therefore it cannot serve as an appropriate modeling tool. Fortunately, the second problem can be solved by replacing GSPN using Finite Population Queuing System

Petri Nets (FPQSPN) [2]. FPQSPN extends Petri net by introducing "shared places and transitions", hence greatly reduces the size of Petri net. Original GSPN only uses exponential distribution on transitions to preserve time independence. However, FPQSPN allows users to use other probability distributions which extend GSPN's stochastic modeling capacity. A Finite Population Queuing Systems Petri Net (FPQSPN) [2] is formally defined using a 9-tuple (***P***, ***T***, *Pre*, *Post*, M_0, s_o, *t*, *tt*, *k*) such that:

- ***P*** is a finite and non-empty set of places,
- ***T*** is a finite and non-empty set of transitions,
- *Pre* is an input function, called precondition matrix of size [|***P***|,|***T***|],
- *Post* is an output function, called post-condition matrix of size [|***P***|,|***T***|],
- M_0: *P* (R) {1, 2, 3, ...} is an initial marking,
- *t*: *T* (R) t R+ is the time associated to transition,
- *tt*: *T* (R) *tt* is the type of time associated to transition
- s_o: Ti :Ti ***T***, Pi: Pi ***P*** (R) {0, 1} determines, whether the place or transition is shared,
- *k*: *k* N is number of customers (in terms of queuing systems, the size of population).

By using abstract service elements instead of well defined WSDL interfaces, an ASCT allows different devices with similar functions (for example, different presentation devices such as monitor, printer, speaker, etc.) be discovered and composed using the same service discovery queries. The approach allows similar services to be considered as alternative candidates. Therefore, ASCT eliminates the problem of insufficient candidate pool at the actual service instance level when applying WS-Pro in pervasive computing. In addition, an ASCT can further reduce the size of the Petri net that models the whole service composition.

Our approach uses abstract descriptions to represent scenarios of the intended composite services. In addition to the informal approach for the ASCT, we support FTQSPN to describe a flow of a composite service in the composition architecture. This gives us a concrete mathematical model to analyze the performance of services running on the proposed architecture. Accordingly, ASCTs are transformed into basic Petri nets for verification of the semantic correctness of the composed service. Note that FTQSPN is an extension of a basic Petri net. Once the ASCT is realized by selecting appropriate actual service instances, we extend this basic Petri to FTQSPN based on their properties. FTQSPN can efficiently depict a sub-diagram representing repeated processes by folding corresponding places or transitions into a non-shared one. Therefore, we can effectively evaluate the service scenario under consideration of the multiple instances of an identical actual service instance. Three distinctive categories of basic ubiquitous services - context providers, context processors and information deliverers are necessary components of an end-to-end composite service. Therefore, an ASCT in Figure 4 should involve at least three different abstract service elements, at least one from each category, to support an end-to-end ubiquitous service.

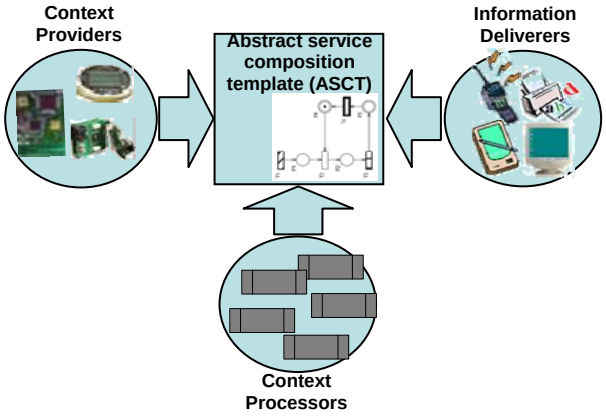


Figure 4: Ubiquitous service composition via ASCTs

4 Preliminary Evaluation

4.1 Enhancement in Reliability and Availability using Virtual Sensor

The Smart Floor [13][15] service deployed in Gator Tech Smart House is a tag-less indoor location system that employs low-cost pressure sensors deployed underneath the residential raised floor to detect the user's current location. As part of the package that offers assistance to mobility-impaired persons, at certain critical locations inside the house such as in front of the bathroom, the user's presence would trigger the smart house to open or close the door. To ensure that the smart floor detects the presence of the user at these critical locations, multiple redundant pressure sensors are deployed.

To evaluate how the virtual sensor framework improves reliability and availability, we simulated a deployment of 7 pressure sensors in four adjacent tiles in front of the bathroom, and created a Singleton Virtual Sensor for each of these physical sensors. A Basic Virtual Sensor was created using these 7 Singleton Virtual Sensors. We assumed that $DQ_T = 70$, a=1 and p = 0.0, 0.6 and 0.8. The different values of p represent the different degrees of similarity in behavior amongst the sensors, which in practical circumstances would depend on factors such as pattern of sensor deployment and the environment where they are operating.

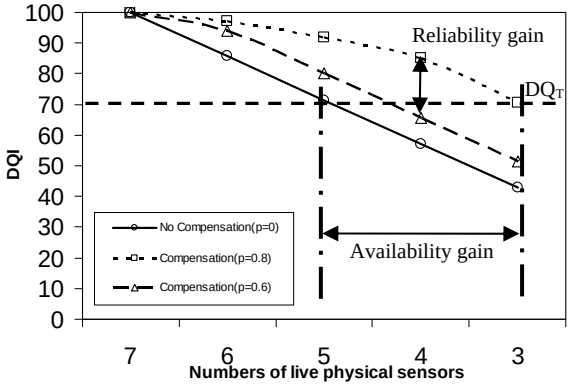


Figure 5: Comparison between different levels of compensation

Assuming all pressure sensors work correctly at the beginning, we simulate a scenario in which every 10 minutes another sensor fails and dies. The result of the simulation is shown in Figure 5. The plot for p = 0 corresponds to the case where no compensation is applied when sensors fail. It can be observed that the compensation algorithm significantly increases the availability and reliability of sensor data in face of sensor failure, even when the number of sensors is small. In this particular scenario, we observe an average gain of 30% in reliability and a gain of 100% in availability when p = 0.8 as compared to when there is no compensation (p = 0).

4.2 Enhancement in Efficient Adaptability using WS-Pro with ASCT

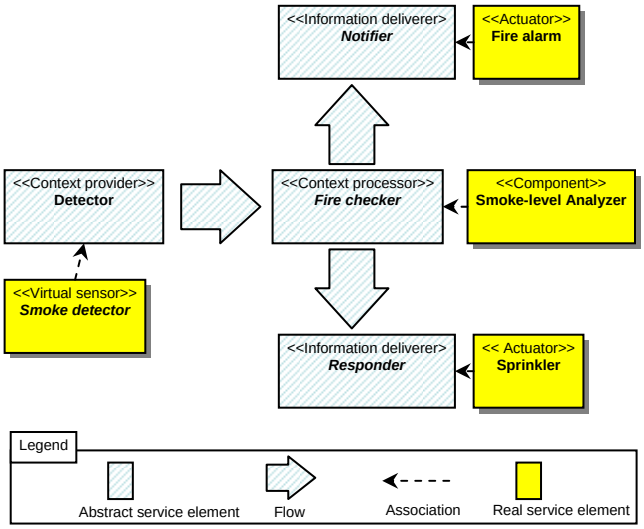
To demonstrate how our WS-Pro/ASCT approach enhances the efficiency in adaptation, let us consider the following scenario. In an efficiency apartment within a smart apartment complex, the digital smoke detector senses unusual amount of smoke in the air, which immediately triggers the fire alarm and initiate the sprinkler system as part of fire emergency service. The fire started by overloaded extension cord, however, quickly spread along the cord and soon engulfed the part of the ceiling where the smoke detector is located. As soon as the smoke detector is gone, the smart house can no longer detects the fire, it shutdowns the sprinkler and it appears that all hope would be lost.

Losing the most direct sensor to detect a fire, the smart house is able to look up an alternative derived virtual sensor for fire phenomena detection. This alternative derived virtual sensor includes an indoor temperature basic virtual sensor as well as a chemical basic virtual sensor. Luckily, the indoor temperature sensor normally used for climate control picked up the unusually high room temperature, while chemical detector finds abnormal amount of CO_2 inside the apartment. Thus, the alternative fire sensors are dynamically created and ready to use for the end-to-end fire emergency management service. WS-Pro/ASCT that supports service re-planning and composition framework is able to substitute a new created context provider with the alternative fire sensor for an original one with the destroyed smoke detector in time. Within a second the sprinkler is back on, and the automatic 911 call to fire department quickly put out a potential tragedy.

We next illustrate how our WS-Pro/ASCT approach models a service composition using ASCT and provides efficient service substitution.

4.2.1 The ASCT for the mission critical service

In Figure 6, an ASCT for an emergency fire management service is associated with a set of actual service instances. Here, the service process for the ASCT consists of abstract service elements denoted with dashed filled boxes and flows denoted with dashed filled arrows. Once appropriate real services denoted with a box are selected by WS-Pro/ASCT, they are associated with an abstract service element. These associations are denoted using shaded arrows




```
<sequence>
 <search name="hazard detector" attribute="fire"/>
 <copy from="virtual sensor type" to "detector type"/>
 <invoke name="fire detector" operation="obtain environment
data"/>
 <search name="hazard processor" attribute="fire" />
 <invoke name="fire checker" operation="identify fire"
attribute="detector type" />
 <search name="emergency service" attribute="fire" />
 <invoke name="fire service" operation="actuator" />
<sequence/>
```

Figure 6: The ASCT for an emergency fire management service.

According to the scenario, the smoke detectors go down quickly. The probe in WS-Pro/ASCT captures this exception and notifies the service composer to do service re-planning. In our case, WS-Pro/ASCT does not need to create a new ASCT. Instead, based on current service process of the ASCT shown in Figure 6, WS-Pro/ASCT should search alternative actual service instances which are still alive, and associate them with the abstract service elements. For example, smoke detector would be replaced with a derived virtual sensor composed of a temperature sensor and a chemical sensor. Similarly, the abstract service element "Responder" would be re-associated with an automatic 911 caller. During this process, WS-Pro/ASCT considers performance data including reliability and availability.

4.2.2 Petri Net Model for the mission critical service in WS-Pro/ASCT

In order to provide timely composition, the Petri net in our WS-Pro/ASCT approach is based on the FPQSPN model. Figure 7 shows the FPQSPN derived from the ASCT illustrated in Figure 6. The object with a double line border represents a shared object such as virtual sensors while the one with a single line border depicts a non-shared object such as a unique software component.

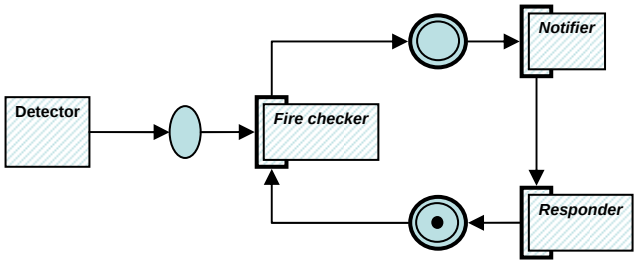


Figure 7: A FPQSPN for an ASCT in Figure 6.

After a FPQSPN model is generated by transforming an ASCT, it can be used to measure the performance of a working mission critical service or evaluate the performance of a new service composed based on ASCT. Note that the real data related to the *t* and *tt* tuples in the FPQSPN model are obtained from the actual service instances associated at present. Multiple instances of the context provider are presented as a non-shared object with the number of the service elements represented as *k* in the FPQSPN model. For this measurement or evaluation purpose, a simulator such as StpnPlay [3] is used. Currently, our WS-Pro/ASCT is planned to have an integrated evaluation module of FPQSPN as part of the metrics computing module.

5 Conclusions and Future Work

While service oriented architecture (SOA) provides a clean interface that allows for uniform programming interface and dynamic self-integration, pervasive computing systems built upon SOA still suffer from failures and dynamic changes in the environment. A comprehensive solution using the concept of virtual sensors to improve the availability and quality of data, and WS-Pro/ASCT - a performance-driven ubiquitous service composition framework - is devised to enhance adaptability and fault-resiliency by efficiently re-planning the end-to-end services in face of component failure or adverse changes. Existing service composition methods for web services employ passive monitoring techniques and provide no performance guarantees of the service composition procedure. The need for a novel approach to compose ubiquitous services is further strengthened by the differences in standard business logic in Web Services and application service logic in ubiquitous services, as well as the fact that capabilities of ubiquitous services are heavily affected by the underlying hardware and communication medium. We are currently seeking a standardized syntax for ASCT, a systematic transformation from ASCT to FPQSPN, and tighter integration of FPQSPN into the WS-Pro framework to ensure seamless integration between them. The other ongoing research concentrates on the validation of this framework, as we plan to use simulation to demonstrate the effectiveness of the WS-Pro/ASCT approach.

We are also conducting research on improving our virtual sensors framework model. This includes the construction of sensor failure models, factoring in the effects of deployment patterns of sensors on reliability and availability and looking at better knowledge representation and management mechanisms for enabling efficient and smart service composability. We also plan on conducting both large scale simulations and smaller scale experiments using a real world test bed of sensors to study the performance of virtual sensors in different settings.

In this paper, we introduced the ASCT and suggested three domain-specific service elements pertinent to end-to-end ubiquitous service composition. From the service development perspective, a service decomposition process is needed in order to identify service elements, including the search of matched virtual sensors, and discover critical ubiquitous services composed of them. Since ubiquitous services pertinent to ubiquitous/pervasive computing systems tend to have common service elements or reflect similar concerns that developer should consider during the system development, this service decomposition process is also required to provide a mechanism to reduce redundancy and capture domain-specific concerns. Hence, one important area of future work involves a service decomposition method to help developers identify necessary service elements, critical services and group redundant or crosscutting concerns. Fortunately, we have a generic software system decomposition method named Function-Class Decomposition with Aspects (FCD-A) [4]. For example, availability and reliability are critical concerns during the development of the service-oriented system to deliver fault-resilient ubiquitous services. The decomposition process of FCD-A can help service developers to identify service elements relevant to them or to classify specific features within them. However, we need to tune the generic FCD-A to support detailed or domain-specific factors required for fault-resilient ubiquitous services. This tuning includes the refinement of the service concepts and the development of new syntax and semantics for FCD-A.

References

[1] L. Baresi and S. Guinea, "Towards Dynamic Monitoring of WS-BPEL Processes", *Proc. Int'l Conference Service-Oriented Computing*, Amsterdam, Netherlands, December, 2005, pp. 269-282.

[2] J. Capek, *Petri Net Simulation of Non-deterministic MAC Layers of Computer Communication Networks*, Ph.D. Thesis, Czech Technical University, 2001.

[3] J. Capek, *STPNPlay: A stochastic Petri nets modeling and simulation tool*, Available at http://dce.felk.cvut.cz/capekj/StpnPlay/index.php.

[4] C. K. Chang and T.-H. Kim, "Distributed Systems Design Using Function-Class Decomposition with Aspects", *Proceeding of the 10th IEEE International Workshop on Future Trends of Distributed Computing Systems (FTDCS 2004)*, Suzhou, China, May 2004, pp. 148-153.

[5] K. A. Hummel, "Enabling the Computer for the 21st Century to Cope with Real-World Conditions: Towards Fault-Tolerant Ubiquitous computing", Keynote speech, *IEEE International Conference on Pervasive Services*, Lyon, France, June, 2006.

[6] H. Ludwig, "Web Services Qos: External SLAs and Internal Policies", *Proc. 4th Int'l Conf. Web Information Systems Engineering Workshops (WISEW'03)*, Rome, Italy, December, 2003, pp. 115-120.

[7] D. W. McCoy and Y. V. Natis, "Service-Oriented Architecture: Mainstream Straight Ahead," *Gartner Research Report*, Available at http://www.gartner.com/pages/story.php.id.3586.s.8.jsp , April 16, 2003.

[8] B. Srivastava and J. Koehler, "Web Service Composition - Current Solutions and Open Problems", *Proc. International Conference on Automated Planning and Scheduling (ICAPS) 2003, workshop on planning of web services*, Trento, Italy, June 2003.

[9] J. Xia, "Qos-based service composition," *Proc. IEEE Int'l Conf. Computer Software and Applications Conference (COMPSAC'06).* student paper, Chicago, IL, September 2006, pp. 359-361.

[10] J. Xia and C. K. Chang, "Performance-driven Service Selection Using Stochastic CPN", *Proc. IEEE 2006 John Vincent Atanasoff International Symposium on Modern Computing (JVA 2006)*, Sofia, Bulgaria, October, 2006, pp. 99-104.

[11] Y. Yamato, Y. Tanaka, and H. Sunaga, "Context-aware ubiquitous service composition technology," *Proc. IFIP International Conference on Research and Practical Issues of Enterprise Information Systems (CONFENIS2006)*, Vienna, Austria, April 2006, pp. 51-61.

[12] L. Zeng, et al, "QoS-aware middleware for web services composition," *IEEE Transactions on Software Engineering*, 30(5), May 2004, pp. 311-327.

[13] A. Helal, W. Mann, H. Elzabadani, J. King, Y. Kaddourah and E. Jansen, "Gator Tech Smart House: A Programmable Pervasive Space", *IEEE Computer*, March 2005, pp 64-74.

[14] R. Bose, A. Helal, V. Sivakumar and S. Lim, "Virtual Sensors for Service Oriented Intelligent Environments," *Proceedings of the Third IASTED International Conference on Advances in Computer Science and Technology*, Phuket, Thailand, April, 2007.

[15] Y. Kaddourah, J. King and A. Helal, "Post-Precision Tradeoffs in Unencumbered Floor-Based Indoor Location Tracking," *Proceedings of the third International Conference On Smart homes and health Telematic (ICOST)*, Sherbrooke, Quebec, Canada, July 2005.