Dependability and Accountability for Context-aware Middleware Systems

Andrew C. Rice
Computer Laboratory,
University Of Cambridge,
15 JJ Thomson Avenue,
Cambridge, UK. CB3 0HL
Andrew.Rice@cl.cam.ac.uk

Alastair R. Beresford
Computer Laboratory,
University Of Cambridge,
15 JJ Thomson Avenue,
Cambridge, UK. CB3 0HL
Alastair.Beresford@cl.cam.ac.uk

Abstract

In this paper we present a framework to provide dependability through accountability. Our proposal exploits the asymmetry present in the majority of sensor data processing to cheaply validate events and processing which occurs at various points in a distributed middleware system. We exemplify our framework with reference to two real-world distributed location middlewares. We adapt one of these to evaluate our framework and present performance results.

1 Introduction

Pervasive computing envisions an era when many heterogenous devices work proactively together. Context-aware computing aims to enable pervasive computing systems to meet these autonomous requirements by using detailed sensor data concerning the state of the user and their environment. Context-aware applications not only share data from sensors, but also higher-level pieces of context information derived from the raw sensor readings. For example, containment data from a location system may be shared by multiple applications; such data are derived from position data of individual entities, which in turn may be estimated from sensors measuring a particular physical property of the environment. One aim of many middleware architectures in pervasive computing is to perform processing steps which may be amenable to sharing between multiple applications.

Many sensor systems are built from a distributed set of machines, whilst end-user applications often run on another set of distributed hosts. We expect wide-scale pervasive computing systems to experience dynamic changes in both the sensor systems available and the population of applications. Therefore, ensuring the correct operation of a pervasive computing system is difficult, yet user confidence in its correctness is vital to the success of pervasive computing—applications which cannot be trusted to operate reliably will not be used. It is important to acknowledge that faults within such a future pervasive system are inevitable. Traditional distributed system failures are, of course, likely, but additional model inaccuracies and environmental assumptions occur in context-aware systems. Furthermore, since pervasive computing devices and applications aim to blend into the environment, it is neither possible, nor desirable, to artifically control the environment in order to prevent failure.

Therefore, as the complexity and heterogeneity of pervasive middleware increases, it is no longer acceptable for an application to simply indicate there is a generic error in the middleware. Providing a detailed analysis of the reason for failure is required by administrators in order to fix the system, and for end-user applications in order to operate with reduced functionality.

In this paper we explore how to build *dependable* pervasive computing applications which utilize a distributed middleware. (By dependable, we mean a system that either performs within specification, or provides suitable feedback to users when faults occur.) We use *accountability* as a mechanism for constructing a dependable system. Accountable applications are capable of describing *why* a particular action was (or was not) taken and therefore accountability helps application writers to acknowledge possible failure modes and to provide feedback both to users and system administrators.

2 Motivation

Software-based techniques such as the recovery block system use *acceptance testing* [5] to validate the results of a software component. If the acceptance test fails for a given block, an alternative block is used to produce the result. Many computations within pervasive computing are amenable to acceptance testing because checking the result of a computation is often computationally much cheaper than computing the correct answer. For example, the Bat system (described in more detail in Section 3) performs multiple iterations of a computationally expensive non-linear regression to find a location from a redundant, noisy data. However, verification of this result by checking that the returned location is consistent with some majority of the distance readings is cheap.

Similarly, vision-based tracking systems such as ARToolKit [3] use image processing techniques to detect fiducial tags and extract their pose and position. This process is necessarily computationally expensive. However, once the tag has been located, checking that it exists in the image in the position specified is relatively cheap. (Our experience at implementing this with our own vision system, Cantag, is discussed further in Section 5.)

Many middlewares for pervasive computing provide a callback mechanism. The SPIRIT middleware [1] supports event-based programming whereby an application registers for a callback when two particular geographic regions surrounding physical objects interact (such as containment). The continual monitoring for a containment event in a large number of arbitrary 3-dimensional regions is a computationally expensive process as compared to the procedure for checking that a located object lies within a particular region. Similar functionality exists in the *IdentityPresence* widget in the Context Toolkit [8].

Most pervasive middleware systems in use today statically distribute processing, or even centralise the majority of computation on a single node. Yet, recent research in active networks, grid computing, and ad-hoc sensor networks has begun to develop more dynamically distributed architectures. We believe pervasive computing will inevitably move towards a dynamically distributed model because: (i) sensors are increasingly attached to mobile devices, leading to a dynamic sensor population; (ii) mobile devices are increasingly used to enable human-computer interaction, leading to a dynamic actuator population; (iii) mobile devices will make use of static computing nodes where possible because they have greater resources; and (iv) the average user of a pervasive computing system will purchase a range of devices at different points in time and expect them to inter-operate seamlessly with minimal configuration.

Validation of results becomes even more critical when dynamic distribution of data processing occurs. In summary, there exist many cases in processing sensor data from context-aware systems where checking the output of the result is cheap when compared with the cost of calculating it. Furthermore, it is possible to exploit this asymmetry to allow impoverished nodes in a distributed computing system to offload and share computation, whilst maintaining confidence in the results through inexpensive checks. The rest of this paper builds on this concept and explores how acceptance testing through inexpensive validation can be used to build accountable pervasive computing systems.

3 Two example middleware systems

We exemplify our concept using two location systems: the Bat system and the Cantag system, which provide markedly different modes of operation, and are briefly explained next.

The Bat system [9] tracks mobile Bats, which transmit an ultrasonic pulse whenever their unique identifier is transmitted over a radio channel. Position information is resolved by measuring the time-of-flight of the ultrasound pulse from the Bat; this timing information is collected by a matrix of ceiling receivers. The Bat system uses a distributed hierarchy of software components, synchronized by a shared, global clock; see Figure 1. Every room within the coverage area contains a chain of ceiling receivers attached to a *matrix manager* (MM). On each clock edge, every MM samples the data from the chain of ceiling receivers and reports any ultrasonic activity as a sequence of distances, based on the time-of-flight measurements made by the ceiling receivers. The MM data are then converted into 3D positions by software running on standard PC hardware—we refer to these nodes as *localizers*. Simultaneously, nodes called *area managers* (AMs) schedule the polling of the mobile Bats and report which Bat has been scheduled for each timeslot. A central node then correlates the location information from the *localizers* and scheduling information from the AMs and produces the location reading.

The Cantag [7] system is a monocular visual tag recognition and location system supporting many tags designs, including those used in the TRIP [4] and Matrix [6] systems. An example Cantag image processing pipeline is shown in Figure 2. The edges shown in the figure denote data flow paths through the system. This system may be run with either a fixed video camera tracking mobile tags (*outside-in* mode) or with a mobile video camera tracking fixed tags (*inside-out* mode). We have also built a distributed version of the Cantag system which can execute different stages of processing on different physical hosts, and thus combine common processing steps used by different context-aware applications via a network.

Both the Bat system and Cantag middleware are distributed systems which produce identity, location, and pose information for the tracked objects. The Bat system deployment covers an area of approximately $500\ \mathrm{m}^2$ whereas the coverage region of an instance of the Cantag system is far smaller (of the order of $5\ \mathrm{m}^2$). Furthermore, when operating in inside-out mode the Cantag system's coverage area varies unpredictably with the movement of the user carrying the camera. A second significant difference between these two systems is that of scheduling; the Bat system is a polled

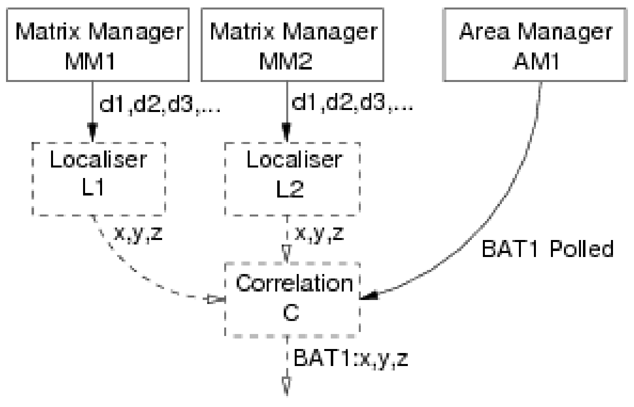


Figure 1. The structure of the Bat system. Solid boxes and lines indicate trusted software components and data.

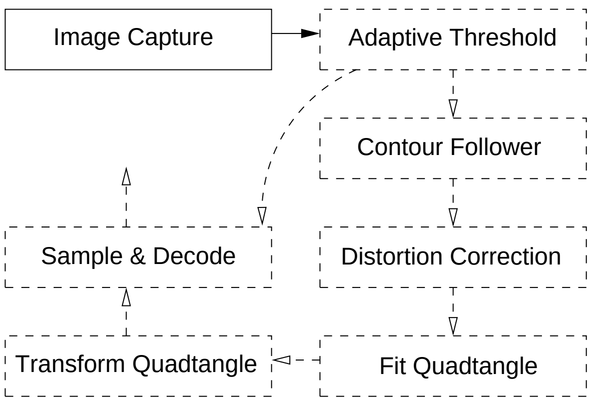


Figure 2. The Cantag processing pipeline for square tags. Arrows indicate data-flow; solid lines indicate trusted components and data.

system: Bats present in the building register with the scheduler and are then polled in turn. From a validation point-of-view this means that we can be sure that the system hasn't been able to track any Bat which is not registered with the system. However, the Cantag system does not know in advance which tags it is looking for in the scene—any valid tag could be displayed to the system and recognised.

4 An analysis of validation techniques

Application-level consistency checks can be used to demonstrate that data produced from middleware components are correct. However, to perform any meaningful validation, we require a set of *trusted* system sensor nodes. Only information produced by trusted nodes are assumed to be correct and dependable, and therefore data from all other software components are considered potentially erroneous. In order to maximise accountability, we wish to minimize the number of trusted nodes. Thus, trusted nodes will usually include only the low-level components of a sensor system. If we are concerned about the presence of malicious hosts or software modules, data from trusted nodes needs to be authenticated using an appropriate certificate system.

In the Cantag system, we might deem the software component that acquires the camera images to be a trusted component. The remainder of the image processing pipeline (which may be executed on several machines) is then untrusted; see Figure 2. The result of the Cantag processing pipeline is a transformation from object co-ordinates to 3D camera co-ordinates. Validation of the existence of a tag, as found by the processing pipeline, does not necessarily require recursive validation of each step within the pipeline—instead an application may check the position and pose of the located tags against the original image. For the Bat system we define the set of trusted nodes to include the MMs and AMs; see Figure 1. An application may then validate the output of the system by checking the location produced against the trusted distance readings from the MM and verify that the Bat reported was actually polled by the AM.

In the remainder of this paper, we define *positive validation* as a mechanism for validating positive (i.e. entity x is at location y) results from a system. Positive validation provides behaviour similar to fail silent hardware: whenever data are produced we can check their validity. However, it is not possible to check that data are not being lost or diverted within the middleware. Some applications are arguably satisfactory with positive validation: for example, in a door entry system we care much more about false positives than false negatives. However positive validation does not address our desire for accountability—the middleware cannot explain why some events (e.g. not opening the door for a valid user) did not occur.

Positive validation does not suffice for stateful system components deployed in the middleware either. For example, consider the scenario where an application deploys a *room monitor* in the Bat system to provide a callback whenever specific Bat enters a particular room. To provide evidence of correct behaviour this monitor must not only demonstrate that the specific Bat is in the chosen room (at some time t) but also that when previously sighted (at time s) the Bat was not in the chosen room—the difficulty here is in establishing that the Bat was not sighted in the period between time s and time t. The situation is worse for unpolled systems: when using the Bat system it is possible to collect a list of all the Bats that have been polled for a number of consecutive time-slots and be certain that no other Bats could have been sighted. However, the Cantag system could find many tags in a single image and so it is much more difficult to demonstrate the complete list of all sightings.

We define *negative validation* to mean the verification of the *lack of* a particular set of sensor readings. Therefore negative validation requires a system to provide evidence concerning the lack of a particular sighting or set of sightings. We have seen (in Section 2) many examples in which positive validation is cheaper to perform than the initial computation to calculate the result. We call such algorithms *positive asymmetric* routines. It is important to note that not all algorithms are cheaper to check than to calculate from scratch; we denote algorithms in this set as *symmetric*. For example, checking the output of the *Threshold* node in the Cantag system requires re-binarization of the source image followed by a comparison against the original output. We note as a curiosity that we have yet to encounter any *negative asymmetric* validation functions.

Many symmetric validation algorithms can be converted into asymmetric validation routines if we accept a probabilistic approach to conformance testing. Looking again at the *Threshold* node in the Cantag system as an example, we can evaluate the result of executing the binarization on only the pixels of particular interest (e.g. the sample points used to decode a tag) or simply randomly select a small subset of pixels to check for coherence with the data provided.

5 Computational costs of validation

We investigated the costs and benefits of validation for an example pipeline in the Cantag system. In this configuration we envisage a trusted node which records the scene and applies the binarization and contour follower steps before offloading the computation into the middleware. This allows applications tracking differently shaped tags to share data already computed at this point. We note that our initial attempts to apply validation to Cantag turned up a number of software bugs in our implementation, demonstrating an additional value of validation as a testing tool!

Our experiences highlight large runtime variations in the computational cost of validation. The time taken to validate an image is highly dependent upon the elements within it. For example, the validation of camera distortion correction must be performed for each contour within the scene. This means that our implementation, which performs the pipeline stage over the whole image before validating it, would perform differently to an implementation which validates each element of the image at a time—these approaches trade-off pressure on the machine's instruction-cache against pressure on the data-cache. We also notice that there are particular stages for which a sufficient validation test is application specific rather than system specific. Consider the transformation stage of the pipeline where a 3D transformation from camera to object co-ordinates is derived from the fitted quadrangle: applications requiring only the binary data stored on the tag (e.g. barcode readers) need only check that the resulting transform preserves the code-plane of the tag. However, applications performing visual overlay must also check that the transformation defines a valid 3D space which is anchored to the matched shape.

To give some insight into the relative costs of positive and negative validation, we executed the Cantag pipeline on a video sequence showing three tags entering and leaving the field of view. A number of *entities* (data structures representing potential candidate tags), are passed between adjacent pipeline stages. Figure 3 shows the number of entities requiring positive and negative validation at each stage. For many stages the additional work required for negative validation is minimal due to the relatively small number of failures. However, for the final stage (decode) the majority of image elements are rejected and require negative validation. This suggests that we may tune the performance of the system by attempting to ensure that rejection of invalid entities occurs in stages with minimal validation costs. The final stage in Figure 3 shows the effect of a heuristic that removes all contours smaller than a certain size—for many frames a significant improvement. This approach applies in general to other systems. Another example, drawn from the Bat System, would be monitoring whether a particular Bat is inside a container, which is itself located inside a particular room—large numbers of sightings can be cheaply eliminated at an early stage because all sighting information from the Bat system is associated with the room that the sensor is in.

6 Conclusion

We have presented a method for accountability which makes it possible for the applications to validate system behaviour. This provides a number of important benefits: (1) users may check the results produced by mobile agents in an untrusted network; (2) applications have explicit checks for system performance, enabling programmers to provide better adaption when the system fails; and (3) system administrators may use the failure or successful validation of invalid events to localize problems in complex distributed systems.

Some system elements (consisting of true asymmetric functions) are highly amenable to validation whereas other elements require significant computational effort. Applications must decide which parts and proportion of its data require validation in order to make the best tradeoff between dependability and efficiency. In some cases, heuristics early in the processing pipeline can be used to reduce the computation cost.

The amount of evidence required to validate any conclusion necessarily increases at each node in the system and

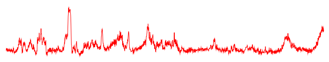
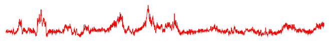
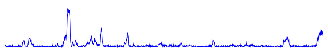
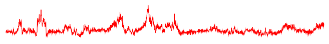
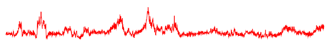
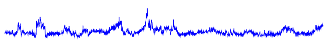
Stage	Positive Validation		Negative Validation	
	entities-per-frame	average	entities-per-frame	average
Distortion Correction		33		0
Fit Quadtangle		28		4
Transform Quadtangle		28		0
Sample		24		4
Decode		2		22
Decode+Heuristic		2		8

Figure 3. The number of entities in each frame of the video sequence is shown graphically. The mean number of entities (a weak description of data with this distribution) is also shown alongside.

so it is inappropriate to transmit evidence with every event. For example, a mobile device making use of data from the Cantag system might request the location of a particular tag whenever it is contained within a particular region. Without accountability the amount of data transmitted to the node is a small number of bytes each time the event occurs. The evidence which accompanies this is a rather more substantial: in order to validate data from the Cantag system the original image must be sent to the mobile which is thousands of bytes of data.

Therefore, nodes in the system must store evidence for posterior validation by applications upon request. We note that evidence may be stored anywhere on the network rather than at the node performing the action.

We have found that validation data can provide a useful tool for debugging a distributed system. Applications might even choose to do no data validation at run-time but instead provide an interface for the end user to request an explanation for a particular action. It has been demonstrated that presenting uncertainty information can improve end-user performance [2]. In future work we hope to investigate whether the system can present validation data to users in order to improve their estimate of system behaviour, especially in highly dynamic networks.

Acknowledgements

The authors would like to thank Alan Mycroft, Andy Hopper, Paula Buttery and Rip Sohan for their insightful comments. This work has been funded by EPSRC.

References

[1] N. Adly, P. Steggles, and A. Harter. SPIRIT: a resource database for mobile users. In *Proceedings of ACM CHI'97 Workshop on Ubiquitous Computing*, Atlanta, Georgia, 1997.

[2] S. Antifakos, A. Schwaninger, and B. Schiele. Evaluating the effects of displaying uncertainty in context-aware applications. In *UbiComp2004: Ubiquitous Computing*, pages 54–69, 2004.

[3] M. Billinghurst and H. Kato. Collaborative mixed reality. In *Proceedings of the First International Symposium on Mixed Reality*, pages 261–284, 1999.

[4] D. L. de Ipiña, P. R. S. Mendoną, and A. Hopper. TRIP: a low-cost vision-based location system for ubiquitous computing. *Personal and Ubiquitous Computing*, 6(3):206–219, May 2002.

[5] B. Randell. System structure for software fault tolerance. In *Proceedings of the international conference on Reliable software*, pages 437–449, 1975.

[6] J. Rekimoto. Matrix: A realtime object identification and registration method for augmented reality. In *Proceedings of Asia Pacific Computer Human Interaction*, pages 63–68, July 1998.

[7] A. C. Rice, A. R. Beresford, and R. K. Harle. Cantag: an open source software toolkit for designing and deploying marker-based vision systems. In *Fourth Annual IEEE International Conference on Pervasive Computer and Communications (PerCom)*, 2006.

[8] D. Salber, A. K. Dey, and G. D. Abowd. The Context Toolkit: Aiding the development of context-enabled applications. In *Conference on Human Factors in Computing Systems*, pages 434–441, 1999.

[9] A. Ward, A. Jones, and A. Hopper. A new location technique for the active office. *IEEE Personal Communications*, 4(5):42–47, Oct. 1997.