

Security and reliability challenges in future consumer devices

Stefano Campadello, Ronan Mac Lavery, Titos Saridakis

Nokia Research Center, Helsinki, Finland

Abstract: Consumer devices are changing from functional hardware to access points for networked services. Following this change the manufacturers must update their development strategies in order to gain the new market. The update of the development strategies includes considerations regarding low cost products (devices plus software infrastructure and end-user applications), flexible and dynamic devices and addressing the needs of service provision. One promising way to address these concerns is the provision of customisable service platforms that are reliable and, from the user's point of view, secure. For the next generation of consumer devices this means software architectures based on components will be used. The discussion below analyses these forces from the device manufacturer's viewpoint, focusing on security and reliability in component based software infrastructures for consumer devices.

Key words: component based software engineering, customisable middleware, embedded systems, software architecture

1. INTRODUCTION

The future consumer devices will no longer be passive pieces of hardware, but will act as a gateway to a large number of services that enhance a user's lifestyle. Following in the pattern of Moore's Law, manufacturers are now producing low cost, pervasive and increasingly connected devices that will permit these types of services. However, for these to materialise there will have to be a commercial-strength software infrastructure to support users, device manufacturers and service providers. One crucial aspect of this future environment is the technical infrastructure linking users and service providers. It is this technological consequence of this future market that needs to be identified and addressed by device manufacturers.

As the user ultimately pays for a service, a key feature of this market will be the support of commercial interaction between users and service providers. Device manufacturers and communication companies must therefore create an enabling infrastructure. This contains access devices and communication links that automate interaction and payment between users and service providers. This overall structure is shown in *Figure 1*. The accessing device could be any commonly used commodity, such as a fridge, microwave oven or television. However, due to the computational and connectivity complexity the first generation will consist of devices that already support some networked service, like televisions, set-top-boxes, PDAs and mobile phones.

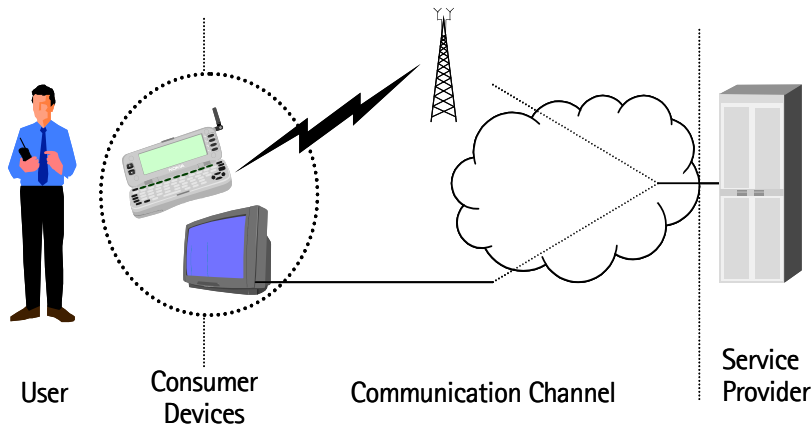


Figure 1. User centric service provision

The services supported by a consumer device can be broadly categorised into those that are supported by remote service providers and those that emerge from increasing connectivity. In an interconnected scenario, users will be able to use one device to interact with another. An example would be the use of a mobile phone to remotely programme the recording of a television programme. Another is the use of the phone to alert a user when their favourite television series is being broadcast. For remotely supported services, the user accesses or is automatically given information from a service provider, who can also act as an intermediary between devices. An example of this would be a continuous weather, television programme, or stock market information sent to a mobile phone combined with a storing and automatic recording service.

Ensuring active participation of users in paying for these services is one key problem that needs to be overcome. Experience indicates that users must trust services before being willing to pay. Among other factors this involves meeting their expectations for security and reliability. Therefore,

consumer device manufacturers must strive to address these prime concerns. Similarly, service providers need to be assured a source of income. Consequently, manufacturers and network operators must provide reliable access points that guarantee service availability.

Manufacturers will also need to support a wide range of services, each tailored for different users. As it is unfeasible for a manufacturer to provide everything for everybody, techniques for providing dynamically configurable consumer devices are necessary. These have to be developed cheaply and rapidly to meet the future demands of the consumer. Current trends point to the extensive use of component based software architecture as a basis for handling product variability in volatile consumer markets.

Some of the necessary requirements for supporting this vision of user centric service provision in consumer devices are identified and discussed here. The emphasis is on challenges to device manufacturers, as the availability of suitable devices is critical for this market. The next section builds on experience from web-based services to predict a user's requirements for participation in paying services. Section 3 describes the system architecture of the next-wave of consumer devices, focusing on the reasons and drivers behind the component based approach to consumer device middleware. The following section discusses the support for security and reliability within this system context. The final section summarizes the discussion and highlights issues requiring research.

2. USER EXPECTATIONS FOR CONSUMER DEVICES

For user centric services to be produced there has to be some method for service providers to fund their activities. This in turn means that device manufacturers have to support paying services which inevitably comes back to the user, who pays for the service provided through the device. While there have been few studies in the area of the economics of consumer devices for paying services, these have tended to focus on technological issues. There is a lack of user behaviour studies on widely deployed consumer devices supporting payment (such as mobile phones). Therefore, to predict the needs of the user for future systems we have used the research based on web-based paying services.

The results of studies into this area have focused on security [1] [2], a key element in the mind of present day users. As such, this research has produced a set of guidelines for web developers. A major requirement is the need to automate security as much as possible while giving the user the feel-

ing of being in control. This involves hiding the intricate details in a trustworthy device.

On the performance aspect of web-services there has been some research into the economic impact on user behaviour [3]. The findings are dramatic in that it appears that small time difference for page download times, for example 7-9secs can result in major changes in user behaviour. Similarly, availability is another issue affecting user behaviour. If a service is unavailable the user will look elsewhere for their needs.

From the device manufacturer the issue of reliability and robustness is important from the perspective of customer satisfaction. If a device fails, the user will naturally blame the manufacturer, but if a service fails will the user blame the service provider, the network operator or the manufacturer? This issue needs to be researched in the future when devices are more widespread, and more paying services are available.

In the aforementioned web-services, the infrastructure used is a standard computer over the Internet. The future networked consumer device will be very different both in terms of appearance and capabilities. Likewise, the user's perception of the device will also be altered. In the current web-based infrastructure, users permit certain degrees of failure, arguing it is a side effect of being on the cutting-edge of technology. In the consumer market, previous expectations regarding reliability dominate. An example of this is the user's expectations of the performance of a mobile phone; it must always work when the network is provided. This applies to all consumer devices, including fridges, televisions and set-top-boxes.

The conclusion drawn is that for networked consumer devices the user expectations will exceed those for the present web-based offerings. Consequently, the infrastructure support for paying services in consumer devices has to cover more demanding requirements than its web-based counterpart has faced.

3. CHALLENGES FOR MANUFACTURERS

While the problems of flexibility and device constraints are important from a technology perspective, the market drivers also need to be taken into account. When designing customisable, component-based middleware for consumable devices, the following forces must be balanced: flexibility, reduced device resources, and user's, but also service provider's, trust and confidence. While reduced device resource requirements are a concern for the developers of components, these are not specifically addressed here; our focus is on the support of security and reliability in such an environment. Flexibility is directly related to the customisability property of the middle-

ware. The user's, and service provider's, trust and confidence in a device and the services it provides are strongly related to the robustness, security and reliability properties of the device. This section discusses the provision of these properties in a component-based software device.

3.1 Robustness and Dependability

The term robustness does not refer to the operation of individual components in the middleware or the application layers but to the quality of smooth and predictable operation of the device and the services available through it, as perceived by the device user. In other words, the user perceived robustness refers to the absence of any device/service behaviour outside what the user may find in the User's Manual of the device and the service's Instructions to Users. This perception of robustness is partially related to the execution of applications and services on the device, and, partially to the execution of a dynamic integration (in terms of download, installation and instantiation) of new components. As the former is related to the application and service development, we focus on the latter in this section.

When the user of a consumer device is using the services offered, they do not want to be preoccupied by intrinsic device or service events. Most importantly, they do not want to experience any inconvenience due to such events. In the context of configurable, component-based middleware, a substantial amount of such events are related to the dynamic integration of new (application and/or middleware) components. The causes for such events may vary from errors occurring during the download, installation or instantiation of a new component, to failures in establishing the appropriate bindings with other components in the device, or to interaction incompatibilities with such other components. Consequently, the user perceived robustness has a direct impact on the dependability properties provided by the component-based middleware.

A representative summary of dependability requirements includes the following:

- problematic component behaviour¹ should be contained in the component where it occurs or, at most, in the services using this component. Requirements in this category map to failure containment techniques.
- dynamic integration of components should be reversible in case it leads to problematic behaviour of some service; in that case, side-products of the dynamic integration should be removed (e.g. resources allocated to the dynamically integrated component which caused the problematic behav-

¹ Problematic component behaviour could be translated to "failure" when an appropriate failure model is defined.

our should be recovered). Requirements in this category map to failure recovery techniques.

- a configuration validation should precede the dynamic integration of a component into a system in order to detect potential problematic behaviour and avoid producing a configuration that leads to it. Requirements in this category map to failure prevention techniques.
- certain services (e.g. those related to security aspects) cannot experience any problematic behaviour; hence the problematic behaviour of components related to such services should be hidden and the services should be delivered despite the occurrence of a problematic behaviour of components. Requirements in this category map to failure masking techniques.

A wide variety of fault tolerance techniques have been suggested in the related literature for dealing with each of the categories of problematic behaviour presented above. However, these techniques must be tailored to the needs and the constraints of consumer devices before they can be applied in the context of component-based middleware as discussed in this paper. For example, resource constraints of certain consumer devices (such as mobile phones) might prohibit failure masking approaches, such as state machine [21] and N-version programming [17].

On the other hand, other categories of consumer devices (e.g. home servers, set-top-boxes, etc) may allow resource consuming approaches to the issues related to problematic behaviour as a result of dynamic integration of new components (e.g. see [15]). Moreover, the above constraints may change depending on the context in which a consumer device operates. For example, a mobile phone with an otherwise limited memory resource can, inside a home environment, use the home server as additional memory or stable storage and hence dependability guarantees that rely on more resource demanding fault tolerance techniques can be provided. Consequently, it becomes obvious that the dependability needs of consumer devices and the services they provide may vary significantly and there is no "single size fits all" solution.

To ensure the user's confidence and trust in the device and its services, the component-based middleware should allow different fault tolerance techniques to be used depending on the service requirements, the device capabilities and the current context. We anticipate that among the most prominent fault tolerance techniques that future consumer devices will use one may find techniques that provide for:

- atomic component integration, which means that the download, installation and integration of new components in a consumer device will have atomic operation semantics. This means the integration of a component will be completed successfully or it would appear as if it was never at-

tempted - in this case side-products of the integration will be removed from the system.

- component replication either for real-time failure masking (e.g. active replication [21]) or for rollback recovery (e.g. passive replication [18]) or for multi-version programming (e.g. N-version approach [17]).
- fail-stop processor semantics [20] for the components, which ensures that the occurrence of a problematic behaviour is contained to the component where it occurs.

In addition, configuration validation (static or dynamic) that will ensure system consistency prior to reconfiguration (e.g. because of component integration or because of the use of a service) might be needed [19].

3.2 Security

In the process of designing middleware for consumer devices security issues get a high priority. In comparison with web-based services, consumers will be less likely to accept security failures that will be otherwise considered as manufacturer faults. For instance, if after downloading a tampered GSM stack from an unauthenticated provider the mobile phone ceases to work correctly the user's complains could be addressed to the device manufacturer rather than to the provider. Furthermore, the middleware shall protect not only the end user but also the service providers; for example, the correct information about the user behaviour has an essential role for billing purposes. On the other hand security features may increase considerably the cost of the device and/or penalise the performance of the service, making them unattractive. A balance between a very expensive but reliable device and a cost-attractive but prone to failure device must be reached by manufacturers.

The minimal set of security requirements such a middleware has to provide is:

1. *Authentication*. Every stakeholder must be authenticated: the user while logging to the device, the device while connecting to a component provider and the provider while answering to the requests of the device. The relationship between device user and service provider can hold only if all the stakeholders keep their identification.
2. *Integrity*. During its life cycle a component must keep its integrity. It must be protected by tampering attempts conducted by the component provider, the user and possibly by third parties during its downloading process. A tampered component can be used to un-

determine the functionality of the whole system, to collect sensitive information or to produce faulty results.

3. *Confidentiality*. The information stored inside a component and the component itself should remain confidential to third party entities. The component itself may be kept confidential by the component provider to avoid unauthorised duplications, while its content could contain sensitive information.

A further guideline is that the user should not be asked to intervene directly to maintain the security features of the middleware [1]; they should be transparent to the user as much as possible. Furthermore the user should be allowed to modify the behaviour of such features if desired.

These requirements, once met, can be merged to support further security requirements. For instance, if a component produces malicious acts, requirements 1 and 2 can be used to investigate the identity and the origin of the threat. Again, if the user refuses to pay for a component they claim not to have downloaded, requirements 1 and 3 can be used to prove otherwise.

If the middleware allows the adding or removing of components the underlying layers of the whole system can be affected. Requirements are needed to protect the operating system, for instance, from the possible tampering of drivers or physical memory shared with other parallel execution environments. This leads to the need of an exact description of the physical requirements (memory, processor power, and battery consumption) and of the predicted behaviour of a component. This description must be made public to the middleware so that a device can a priori evaluate if it will fit the system once downloaded.

Furthermore, the level of security of a system is represented by the security of its layers. Looking at *Figure 2*, the middleware depends upon the underlying layers, in particular upon the communication channels. As it is not possible to predict which communication protocols will be available making assumptions about the security level of the link impossible. On the other side, the applications running on the top of the middleware depend on its security level. This intrinsic web of relationships is the biggest security challenge manufacturers of future devices are facing today.

Many protocols and tools exist out-of-the-box to address the previous requirements [5], but they have specific requirements and constraints. It will be the duty of the middleware designer to choose the

right combinations of them, considering the needs of the future consumer devices.

4. FUTURE SOFTWARE SYSTEMS FOR CONSUMER DEVICES

Against this backdrop of increasing demand for dependable consumer devices for paying services is the need to support the related applications. For the domestic market these applications are highly dependent on the user and the access device. For example, a weather forecast service depends on the level of knowledge of the user; one might be able to interpret isobar charts, while another will rely on a forecaster's view of the coming weather. The same service might appear very differently on a mobile phone than on a television or web-tablet.

For consumer device manufacturers to succeed in this marketplace, they must aim to reach the maximum available number of users possible. This will allow them to benefit from mass production of similar devices, provided these devices can respond to the user's needs. Therefore, they need to develop platforms that are readily adaptable and can support a large number of applications.

The need to reduce costs, increase functionality and flexibility are key drivers for success. Furthermore, another difficulty arises inside this vision of the future market; *the device manufacturer cannot know all the applications the user will desire*. At best, the general functionality will be known, but the user will define the end application format and functional requirements.

To cope with this problem, device manufacturers need to define open software platforms for their hardware, so that third party applications can be developed independently for different users. This effectively sub-contracts the generation of applications so that the manufacturer can concentrate on the problems and issues associated with device production. Allowing independent application development also permits localisation for different cultures, specialisation for different services and can, with the appropriate open platform, reduce time-to-market.

4.1 System Architecture

The current approach to providing these types of platforms is to build a layered architecture as shown in *Figure 2*. This is split into three basic levels, hardware, middleware and applications. From the software perspective

this can again be split into more levels, hardware abstraction, middleware support, application logic and UI layers.

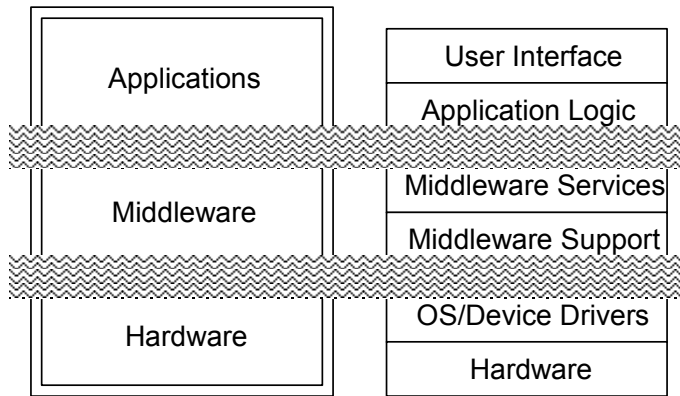


Figure 2. System architecture layers

The above picture also represents a scale of change frequency. The underlying hardware may remain the same for a number of product generations (or versions, or variants) of some category of consumer device (e.g. mobile phone), as may the OS layer. However, successive generations of a consumer device may contain different versions of middleware support, and, even more likely, different (enhanced or richer) middleware services. Finally, applications running on a consumer device may differ even within the same generation of the device. To support this type of flexibility inside the system it has to be incorporated inside each of these layers. Furthermore, to decrease time to market each of these layers has to be reused at much as possible

An example of this process is the use of the same middleware and applications in an entire range of mobile phones produced by the same manufacturer. As high-end phones provide more capabilities in terms of hardware than low-end phones, the middleware layer can be enhanced/enriched to provide facilities and services based on the additional hardware capabilities. This is paralleled by more advanced applications and user interface functionality support in the upper layers. If the enhancement of the middleware and the applications is based on the reusing parts of the middleware and the applications that exist for the low-end phones, the manufacturer saves in costs, time-to-market, robustness and maintenance.

To extend or adapt the contents of a given layer in an orderly fashion it must integrate new functionality without affecting the existing system. This requires a layer software architecture that supports both extension and adaptation. This exists in a trade-off with the need to optimise everything for the

device, such as power consumption, memory usage, processor requirements etc. To achieve a global optimum a holistic approach is necessary, where the distribution of functionality across the layers results in the lowest possible system requirements.

A common process within the software industry is the migration of useful properties and features from the edges of software architectures to the core elements [4]. In the consumer devices this means placing as much as possible in the middleware to support the services required by applications, and to make it flexible enough to adapt to new hardware. Increasing the functionality of the middleware has its drawbacks, most notably a corresponding increase in size and complexity, both of which have dramatic impact on consumer device software. The increase in size is matched by an increase in required resources, which in turn increases the cost of device and reduces potential profits. More complexity means that the cost of producing dependable software increases. Consequently, the middleware layer must be customisable in order to allow both the reuse of its parts and the optimal configuration of its constituents

One way to counter balance the needs of future systems is to build the middleware out of reusable software components. These can be introduced into the system, statically at build time, or dynamically at run-time. In this scenario, only those middleware services that are required by an application are included in the running system; this reduces the resource requirements. If the dynamic approach is taken, introducing the necessary components in response to application needs can support large numbers of applications. This allows the adaptation of the final device to the end-user's needs. This approach is illustrated in *Figure 3*.

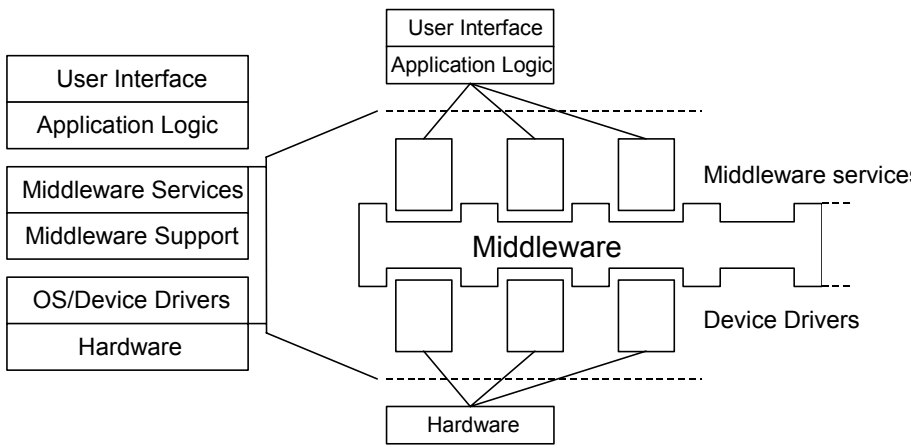


Figure 3. Component based Middleware

4.2 Component based middleware

While this solves the problems of flexibility and resource requirements other problems arise. Most notably, how to provide the flexible middleware layer, how to dynamically adapt the system based on application requirements and how are application requirements expressed? The approach taken by the OMG, Sun and Microsoft in large-scale system is to use components to encapsulate reusable, flexibly deployable software elements. These are combined at build-time or run-time with application to construct end-user systems.

There is some disagreement about the definition of a software component, so for this paper the component definition is taken from [6]:

A component is an opaque implementation of functionality that conforms to a component model and is composable by third parties.

By implication from this definition:

A component is the unit of replacement in a software system.

A component model defines the permitted interaction between components, how component types are defined and how the functionality is made opaque to a component client. Middleware components are therefore the primary unit of deployment and adaptation for the middleware services required by applications.

An example of this approach can be based on the results of the EU ITEA ROBOCOP [11] project. This seeks to define a component system for constructing consumer device middleware. In this context middleware is the software that resides inside a system between the applications and the underlying platform (the OS, device drivers and hardware). The project aims to produce a system that is resource efficient and that can support secure and robust operation. Furthermore, the need for more dynamic consumer systems means that component download, upgrade and extension need to be supported. All these requirements must be resolved inside a single device, ranging from set-top-boxes to mobile phones. As a major aim of this project is to enable a market for middleware component, effort has been taken to ensure that components can be easily reused on completely different hardware/software platforms. The *Figure 4* shows a layered view of the component architecture.

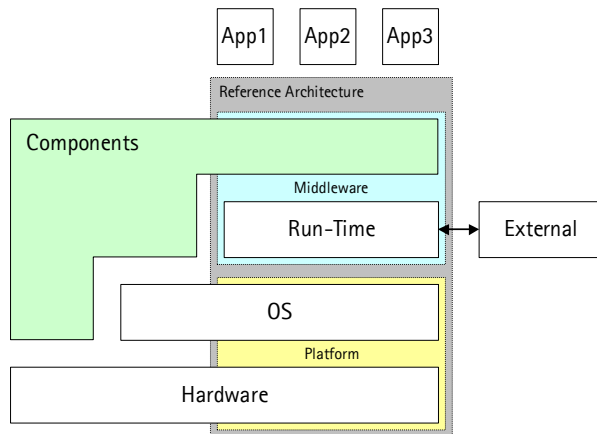


Figure 4. Component architecture

The project's first step was to define a component model for the entire system. This defines the basic interaction style supported and a reference model for components. This reference model is loosely based on COM, with some added elements from CORBA and Java Beans. One key addition, based on industrial experience in KOALA, is the introduction of optionally required dependencies, which can be used to support component configuration. The component lifecycle has been carefully defined; including points where security and robustness checks can be implemented. There is also a defined resource management framework, which uses quality parameters to define a component's resource requirements. This is also integrated into the overall ROBOCOP framework, through clear role separation and defined interaction points between resource controllers, components and the infrastructure. Provided components conform to the same interfaces the design of the system reduces the possibilities of functional mismatch. It also allows components to negotiate for their operational resources; providing users of resource aware components to be assured of a certain quality of service.

5. RELATED WORK

Once the above approach has been taken, the focus moves to providing component support for consumer devices given the tight resource constraints. To date there has only been a few successful attempts at generating component-based consumer device software, such as KOALA [7]. These systems tend to be proprietary and result from a business's own needs to develop software rapidly and efficiently. In our vision of the future this will

have to change to allow increasing numbers of application developers to fill end-user niche needs.

Of the open commercial component systems available, all of these have been designed with a client server model, with the assumption that a server can be purchased with enough memory and processing power to meet the system requirements. This is contrary to the needs of the consumer device market where the goal is to provide a service with the minimum device resource requirements. Subsequently, COM [8], CORBA [9] and EJB [10] in their present versions are unsuitable for this market. This fact has been realised by both the OMG and Microsoft, who have developed reduced versions of their systems, minimum CORBA and WindowsCE. However, both these are designed to be compatible with their current large-scale architecture and have to support compatible architectural assumptions.

On the other hand, the ROBOCOP project concentrates on the dynamic aspect of middleware necessary to provide upgrades and to support new applications. This involves the insertion of new components into the middleware during run-time and raises a large number of new issues. One essential criterion for the middleware is automated downloading of new software into the system. This has to be achieved in a robust, efficient and secure manner. Once obtained, the component has to be integrated into the running system and released to applications. Part of the integration of the component is the realigning and balancing of the system to achieve an optimum configuration with respect to resource consumption and performance.

Adapting the system in response to the introduction of new functionality is still an area of active research. Some researchers have chosen a passive mode of middleware operation, where the applications demand components, resulting in a reconfiguration of the middleware to support their inclusion [12]. Other research has taken a more ambitious, reflexive approach to permit applications to reconfigure the middleware. This also allows the middleware to interact with and configure the applications to achieve a globally optimum system configuration [13]. The potential advantage of this method is the possibility to optimise the overall system in response to individual application needs. However, it is difficult to ensure system consistency using this approach, so certain constraints have to be placed on the adaptation mechanisms used [14]. At this stage, we consider this technique too immature for inclusion in the first generation of the next-wave of consumer devices. We believe these will be based on more the straightforward application of dynamic component-based approaches inside the middleware.

6. CONCLUSION

Given the current status of the development of software architectures for future consumer devices, the above discussion points to several different areas requiring research. The first involves a more detailed definition of the methods and technical requirements needed to support security and reliability in resource constrained devices. A second area of research focuses on the interplay between reliability and security inside a system. The final area of research builds on the previous ones and creates an iterative approach to this work; it involves user studies based on the provision of real services. This combines all the required elements and can be used to generate more concrete and precise requirements for consumer device manufacturers.

The broad categories of system requirements presented in section 3 cover many different concerns of the security and dependability aspects of future consumer devices. However, all consumer devices will not address all these concerns. More likely, different types of such devices (e.g. home server, fridges and mobile phones) will address a different subsets of these requirements. The differentiation factors depend heavily on the user expectations from a device and the services provided by it, the capabilities of the device itself, and the contractual obligations of the services offered by a device. For example, resource limitations combined with low liabilities by service providers in case of an unexpected service behaviour will significantly restrict the dependability and security guarantees provided by a given device (e.g. accessing a weather forecast service from a mobile phone is unlikely to be highly dependable or secure). On the other hand, the role of the device in its user's everyday life will strongly influence the dependability and security requirements on the device and the services it provides (e.g. a home server responsible for physical and electronic access to a domestic environment cannot afford any failures or security holes).

Dealing with both security and dependability is known to be notoriously difficult [22]. On the other hand, the combined impact of security and dependability requirements on the system design has to be taken into consideration. For example, a corrupted message (communication failure) can be interpreted by the receiver as a security threat (malicious behaviour of the sender). Vice-versa, a malicious component exhibiting arbitrary behaviour may cause a byzantine failure detection mechanism to label it as a failed component. Similarly, the interception of a corrupted message by a security mechanism might lead to the isolation of the sender in the system configuration, instead of leading to a negative acknowledgement for the corrupted message. Likewise, the detection of a malicious component as a failed one might lead to attempts to restart the component instead of removing it from the system configuration. The bottom line is that the analysis of the depend-

ability and security requirements must be done simultaneously to avoid designing a system exhibits inconsistent behaviour with respect to these requirements and the system designer's intentions.

The final research area involves creating services that involve wide spread user interaction. These can be used to generate information about user expectations and behaviours when using consumer devices to access services. It is when the user of a consumer device interacts with these types of services confidently that the challenges currently facing device manufacturer will have been met.

7. REFERENCES

- [1] K. Karvonen, *Creating Trust*, Proceedings of the fourth Nordic Workshop on Secure IT systems (Nordsec'99), November 1-2, 1999, Kista, Sweden
- [2] A. Basso, D. Goldberg, S. Greenspan and D. *First impressions: emotional and cognitive factors underlying judgments of trust e-commerce*, Proceedings of the 3rd ACM conference on Electronic Commerce, pages 137-143, 2001.
- [3] Zona Research Inc., *The Economic Impacts of Unacceptable Web-site Download Speeds*, White Paper, April 1999, <http://www.zonaresearch.com/>
- [4] J. Bosch, *Design and Use of Software Architectures*, Addison-Wesley , 2000
- [5] Bruce Schneier: *Applied Cryptography: Protocols, Algorithms, and Source Code in C*, John Wiley & Sons, 2nd Edition, 1995
- [6] F. Bachmann et al, *Volume II: Technical Concepts of Component-Based Software Engineering*, Technical Report CMU/SEI-2000-TR-008
- [7] R. van Ommering, F. van der Linden, J. Kramer, and J. Magee, *The Koala component model for consumer electronics software*, IEEE Computer , Volume: 33 Issue: 3 , March 2000
- [8] *The Component Object Model: A Technical Overview*, <http://msdn.microsoft.com/library/>
- [9] The Object Management Group, <http://www.omg.org>
- [10] Enterprise JavaBeans Technology, <http://java.sun.com/products/ejb/>
- [11] Robust Open Component Based Software Architecture for Configurable Devices Project, <http://www.extra.research.philips.com/euprojects/robocop/>
- [12] F. Kon and R. Campbell, *Supporting Automatic Configuration of Component-Based Distributed Systems*. 5th USENIX Conference on Object-Oriented Technologies and Systems (COOTS'99), pp. 175-187. San Diego, CA. May 3-7, 1999
- [13] G. Blair et al, *The Design and Implementation of OpenORB V*, IEEE Distributed Systems Online Journal, Vol. 2, N. 6, 2001
- [14] N. Parlavantzas, G. Coulson, M. Clarke, and G. Blair, *Towards a Reflective Component Based Middleware Architecture*, in Workshop on Reflection and Metalevel Architectures, June 13, 2000, Sophia Antipolis and Cannes, France.
- [15] L. Sha. *Dependable System Upgrade*. Proceedings of the 19th IEEE Real-Time Systems Symposium, pages 440-448, 1998.
- [16] M. Astley, D. C. Sturman, and G. Agha. *Customizable Middleware for Modular Distributed Software*. Communications of the ACM, 44(5):99-107, 2001.
- [17] A. Avizienis. *The N-version Approach to Fault-Tolerant Software*. IEEE Transactions on Software Engineering, 11(12):1491-1501, 1985.

- [18] N. Budhiraja, K. Marzullo, F. B. Schneider and S. Toueg. *The Primary-Backup Approach*. In S. Mullender, Distributed Systems, 2nd edition, 1993.
- [19] P. Feiler and J. Li. *Consistency in Dynamic Reconfiguration*. Proceedings of the 4th International Conference on Configurable Distributed Systems, pages 189-196, 1998.
- [20] R. D. Schlichting and F. B. Schneider. *Fail-Stop Processors: an Approach to Designing Fault-Tolerant Computing Systems*. ACM Transactions on Computing Systems, 1(3):222-238, 1983.
- [21] F. B. Schneider. *Implementing Fault-Tolerant Services Using the State Machine Approach: a Tutorial*. ACM Computing Surveys, 22(4): 299-319, 1990.
- [22] J. C. Laprie, editor. *Dependability: Basic Concepts and Terminology*, volume 5 of Dependable Computing and Fault-Tolerant Systems. Springer-Verlag, 1992.