T U M

I N S T I T U T   F Ü R   I N F O R M A T I K

SBSE'03 Service-Based Software Engineering Proceedings of the FM2003 Workshop

Ingolf Krüger, Bernhard Schätz, Manfred Broy, Heinrich Hussmann (eds.)

TUM-I0315
September 03

T E C H N I S C H E   U N I V E R S I T Ä T   M Ü N C H E N

FM’03 Workshop on Service-Based Software Engineering (SBSE)

Manfred Broy[1], Heinrich Hussmann[2], Ingolf Krüger[3], Bernhard Schätz[1]

[1]Fakultät für Informatik, Technische Universität München, 85748 Garching, Germany
[2]Fakultät für Informatik, Ludwig Maximilians Universität München, 80538 Munich, Germany
[3]Computer Science & Engineering Department, UCSD, La Jolla, CA 92093-0114, USA

Service-based systems engineering has proven useful in the development of telecommunication systems, helping to modularize complex system functionality with high degree of interaction between system components. Increasingly, the notion of service is gaining ground in other application domains like spontaneous networks, ubiquitous computing, and safety critical systems from the automotive or avionics domain. Precise specification and correct implementation of requirements for services are essential in most of these application domains. Jini, .NET, and SOAP are examples of recently proposed middleware technologies in which syntactic service notions play an important role as an implementation concept.

Surprisingly, however, no precise mathematical foundation supporting the use of services across development phases exists to date. Many notions of service, like those used in the middleware technologies mentioned above, refer only to syntactic interfaces. This is inadequate for more elaborate service specifications that include, for instance, Quality-of-Service properties. Consequently, services are not treated as first-class modeling elements, say, in UML/UML-RT and SDL. Therefore, especially in the application domains mentioned above, a suitable notion of service is needed to support service-based software engineering beyond simplistic syntactic approaches.

A solid methodological basis for service-oriented system development requires a suitable notion of service and the availability of systematic modeling, development, and quality assurance methods based on this service notion.

Therefore, the topics addressed in this workshop include:

- Formal foundations for services
- Modeling notations for services
- Composition operators for services
- Quality-of-Service specifications
- Patterns for service development
- Methodologies for service-oriented system engineering
- Refinement and refactoring techniques for services
- Service-oriented validation and verification techniques

The goal of the workshop is to bring together researchers and practitioners interested in service- oriented software and systems engineering, with an emphasis on:

- using services to describe complex system functionality
- integrating service descriptions to form a system specification
- constructing architectures supporting a service-based development process

to further the definition and introduction of a precise as well as applicable notion of services.

Services as a means of specifying system functionality in a modular fashion are situated at the intersection of formal methods and applied systems engineering. An applicable notion of service requires abstract as well as precise models and notations. Here, proven approaches from formal methods can contribute to supply those models and notations, and to explain how services can be combined, refined, implemented, or deployed.

The unique positioning of services within the landscape of development tasks and techniques is reflected by the contributions selected for this workshop. In “Modelling Electronic Service Systems Using UML” Skene, Piccinelli, and Stearns present a UML profile for service specifications; it relates business workflows with the electronic infrastructure on which they are executed, resulting in the definition and foundation of electronic service systems. Ribeiro, Rosa1, and Cunha1 present and formalize an architecture for Quality-of-Service modeling and deployment in their paper “User Quality of Perception: Towards a Model for Personalised Communication Services”. The problem of service interaction (sometimes also called feature interaction) is addressed by means of tailored diagrams and corresponding analysis algorithms in the paper “Automatic detection of service interactions from graphical specifications”, by Jouve, Le Gall, and Coudert. “Service Discovery in Mob$_{adtl}$”, by Montangero, Semini, and Semprini, addresses the formalization of service discovery in distributed systems, an increasingly important aspect not only of mobile and ad-hoc communication networks.

In addition to these contributions, the workshop proceedings include two invited presentations: In “Automotive Infotronics: An Emerging Domain for Service-Based Architecture” Nelson and Prasad describe a service-oriented approach to automotive software development, tackling the rapidly increasing complexity of software solutions in this application domain. In their contribution “ServiceFORGE: A Software Architecture for Power and Quality Aware Services” Cornea, Dutt, Gupta, Mohapatra, Nicolau, Pereira, Shukla, and Venkatasubramanian present an approach to dealing with Quality-of-Service aspects, including power-management, in distributed and embedded system architectures.

We are also honored to include an invited talk by Pamela Zave, a true pioneer in the domain of service-based software architectures and development methods, in our program.

The organizers are very grateful to the authors of the papers and talks presented at the workshop for their submissions. We also thank the members of the program committee for their efforts in selecting the contributions for this volume, as well as the organizers of FM’03 for their support in realizing this workshop.

We wish all participants a very interesting and successful event!

Workshop "Service-Based Software Engineering" Pisa, September 8th, 2003

Table Of Contents:

Automotive Infotronics: An Emerging Domain for Service-Based Architecture

E. C. Nelson, K. V. Prasad

Ford Research and Advanced Engineering, Dearborn Michigan, U.S.A
enelson7@ford.com, kprasad@ford.com

Abstract. The increasing complexity of automotive electronics systems has led to the evolution of distributed multi-network systems within a vehicle. The features supported by these systems are increasingly dependent on the interactions of distinct components designed by different suppliers. The traditional top-down systems engineering approach to vehicle electronics software integrates application software into individual components directly on top of the low layer hardware support code. Because of the increasing level of interaction between different components, this approach is no longer adequate. As a result automakers are moving towards defining a middle layer of software that is organized in terms of services. Defining the semantics of the services in this layer is one of the major challenges that must be resolved to carry this approach through.

Introduction

The complexity of automotive electronic systems is increasing rapidly. Today's cars have 20 to 30 processors in them, with as many as 70 in fully equipped luxury vehicles. In some vehicles the entertainment system alone uses nearly 20 processors. Some of the processors in a vehicle perform tasks that are very simple, such as controlling a retractable antenna, or the timing of an interval windshield wiper motor, while others perform tasks that are computationally demanding, such as route calculation for a navigation system. The various processors are typically connected with other units performing related functions by digital local area networks. A vehicle may have three or more networks connecting modules that make up different subsystems within the vehicle. For example there might be separate networks for powertrain control, for modules that control various vehicle and chassis functions such as door locks lights instrument cluster etc., for control of safety systems such as airbags and for entertainment and multimedia systems. Each of these networks will contain modules manufactured by a number of different suppliers. Moreover, the modules that are present on a given network differ from vehicle line to vehicle line and within a vehicle line depending on the particular set of options installed in a given vehicle. For example, the same car model may have one of two different radios depending on the audio options chosen by the customer.

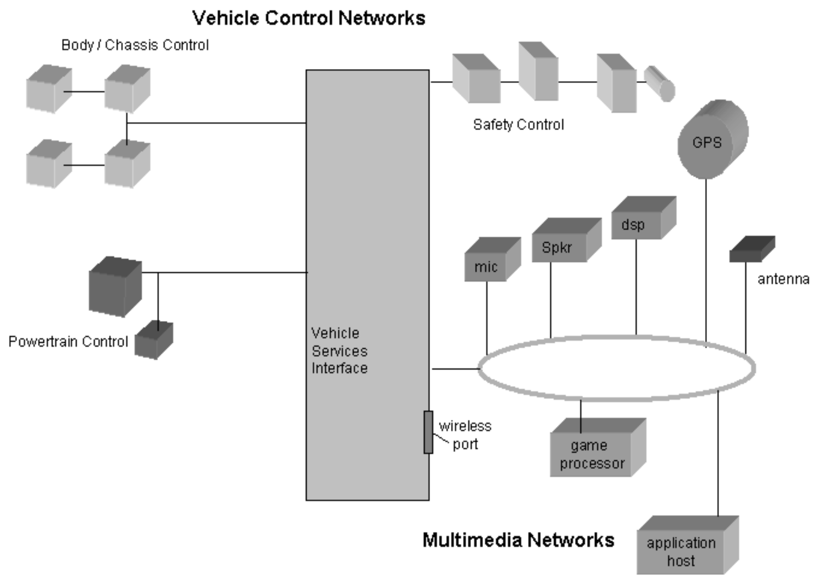


Fig. 1. In-vehicle local area networks

Although the individual networks in a vehicle are largely independent, there is an increasing trend towards interaction between modules on different networks. For example, a navigation module on an infotainment network may need information from wheel speed sensors on a body network to perform dead reckoning position calculations. Or an audio amplifier on the entertainment network may use information from a door module and from the vehicle speed sensor on the body network to automatically raise the volume when the windows are open and the vehicle is moving above a certain speed. Vehicle electronics systems are thus increasingly becoming distributed computing systems where the correct operation of any feature implemented by one part of the system depends on calculations done in another part of the system. Moreover, the nodes in this system are heterogeneous and are all designed by different suppliers. For example, a vehicle might contain a rear seat family entertainment center from one supplier, a front audio head unit from another supplier, and a navigation system from a third supplier, all of which must share components such as a CD player, speakers, and so on. These components must interact over a common control network. Such interaction requires that the behavior of each module be adapted to work with the other modules in the system. The standard top-down design approach leads to rapid increase in design complexity, since the number of behaviors of the system goes up as the product of the number of behaviors of the components.

Even a relatively simple function like a door lock control interacts with a large number of other modules. The diagram below shows a simplified view of the associations between a central locking facility and other objects on a vehicle that has remote keyless entry (RKE) feature which illuminates the vehicle's interior lights when the vehicle is unlocked. The feature also sets driver preferences for seat position, radio presets and mirror position according to which key fob or mechanical key is used to unlock the car as well as flashing the exterior lights to confirm that the vehicle is locked, when the driver presses the lock button on the RKE from outside of the car. The central locking facility must also receive information from the crash detection system to meet the regulatory requirement that the doors be unlocked in a crash, from the vehicle speed sensor and the ignition key status to implement lock on drive away, and from the door ajar sensors.

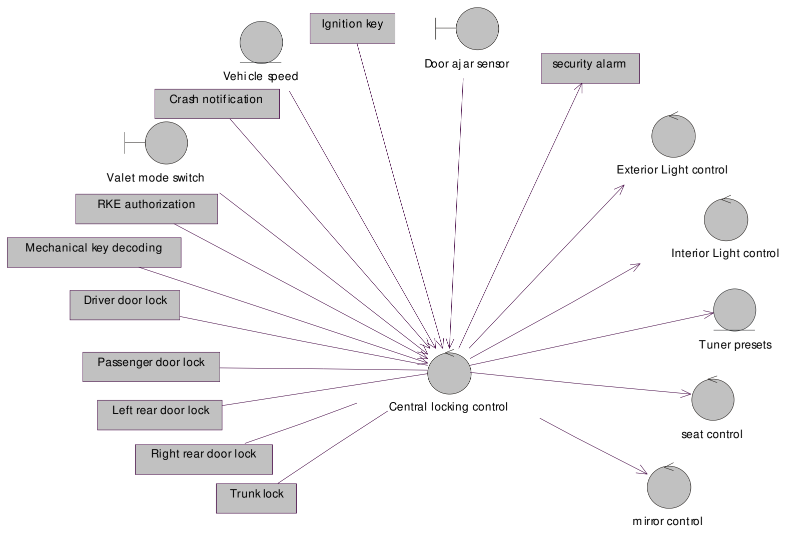


Fig. 2. The interactions of the central locking system in a vehicle

The software entities shown in Fig. 2 are apportioned to physical modules in a way that is heavily dependent on physical packaging and wiring considerations. For example, the doors each contain a module that has the door lock control and latch sensing mechanisms and may contain the door ajar sensor, the mechanical key decoding mechanism and other functions as well, such as power window control.

To illustrate the complexity of the interactions involved consider the problem of generating a door lock acknowledgement. When the driver exits the car and presses the lock button on the key fob, the vehicle will flash the exterior lights to acknowledge that the car is locked. In order to do this correctly, the central locking facility must not only check for an acknowledgement of the lock request for each door module, but it must also check the door ajar sensor for each door to ensure that the door is actually closed. A sequence diagram for this operation is shown in Fig. 3.

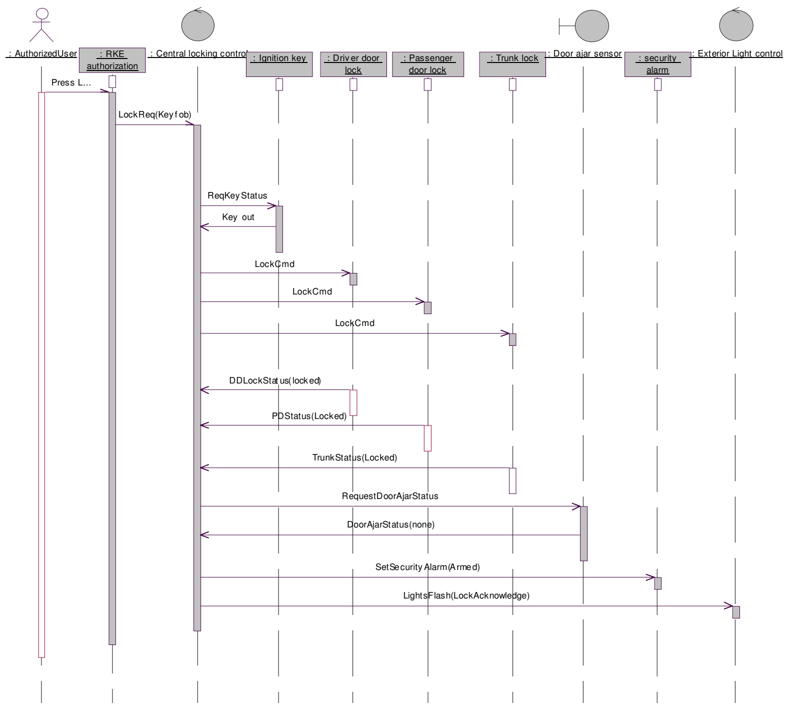


Fig. 3. Sequence diagram for lock acknowledgement

This diagram appears to be straightforward, but the design presented here overlooks an important condition. If the vehicle's occupants leave one of the windows down when exiting the vehicle, the central locking mechanism may or may not acknowledge the vehicle as "locked" depending on how the door module, which typically contains both the lock and window controllers, is specified. The door module lock controller might refrain from returning a "locked" status if a window is open, but this could cause problems in other situations where it is acceptable for the window to be open when the door is locked (drive away lock, for example). Additionally, the solution does not work if for some reason the window control and door lock control are allocated to different modules. The problem of managing issues such as this becomes intractable when one tries to reuse modules across vehicle models, because the semantics of the LockStatus message depend on the module that implements it. In order to keep the acknowledgement policy where it belongs – in the central locking object – the window closure status needs to be available as a service that can be used by the central locking facility in the same way that it uses the door ajar status.

Development of Automotive Electronics

Before we proceed to a discussion of service-based architectures in the context of automotive infotronics, a brief overview of the progress of automotive electrical, electronics and software technologies will help place this paper in context. The history of these technologies placed against a 100-year timeline, as illustrated in Fig. 4, reveals some noteworthy trends: Every few decades there is a major *disruptive* technology enabler: First there were electrical technologies, for instance, the magneto ignition system in the year 1902 [1]. Then came the era of pre-semiconductor (vacuum tube) electronics: There were wireless phones in police fleets in the 1920s [2] and radios [3] in the 1930s. With the advent of semiconductors in the 1950s and 1960s came compact radios. In the late 1970s, the era of programmable semiconductors began, resulting in the early electronics engine control modules. This was in many respects the beginning of in-vehicle software. Software of course played a relatively modest role, delivering one major function --- such as being able to control the engine electronically and one associated feature, such as being able to control tailpipe emissions to desired (or government mandated) levels. The next major step was the development of network communication systems in vehicles [4]. This coupled with the advent of complex "radios" with AM/FM/CD/DVD and navigation features began to demonstrate the unquestionable increase in software enabled functions and associated features in automobiles. The need for model based software and system engineering began to take root in the mid 1990s and today software modeling and associated "autocode" generation is globally accepted within the automotive industry [5]. Over the past thirty years the role of software in automobiles has shifted from a minimal presence in a silicon-defined module to a major controlling presence in most electrical & electronic systems. Whether it is today's powertrain and engine con troll modules or navigation systems, software and associated modeling impacts the look, touch and feel of automobiles. What about the future of automobile software? In many respects, examining today's technologies in the fast-cycle industries –the consumer electronics, the computing and telecommunications industry sectors, may reveal trends in automobile-software. As we look to the future, it appears that software will begin to increasingly define the functionality of features and functions. Today we are witnessing the emergence of software-defined radios [6] and it is not inconceivable that in the future, may of today's silicon-defined functions in the automobile will indeed be replaced by software-defined functions with silicon playing a secondary role to software (see Fig. 5). Concurrent with increasing software-defined functionality, one other potentially disruptive technology is reconfigurable silicon [7] --- a technology that is just emerging.

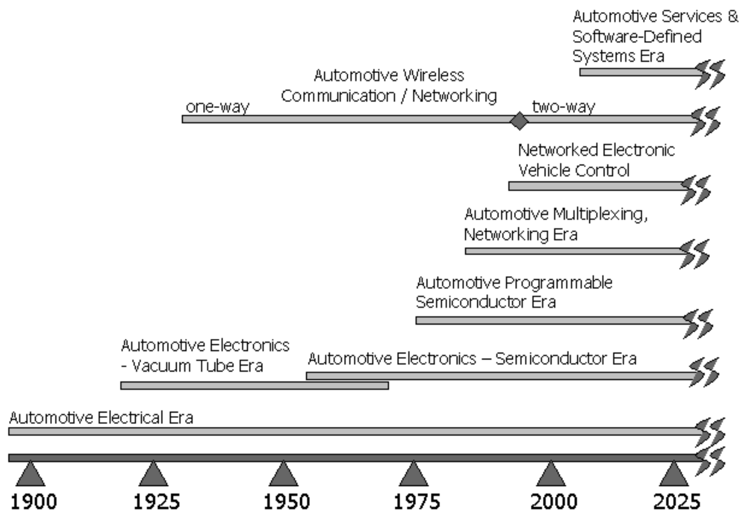


Fig. 4. The Progress of Electrical, Electronics and Software Technologies

As indicated in Fig. 5, the electronics era formed the foundation for the mechatronics [8] era. With the rapid growth in information technologies and associated frameworks for design and development of information systems the automotive industry recognized [9] the need for an integrated approach toward combining information (technologies/systems) with mechatronics and electronic systems --- where appropriate for the functions and features being supported. The result was the creation of the field of automotive infotronics. Simply stated, infotronics systems are designed and developed on the premise that the associated subsystems and possibly components are inter-connected more on the basis of the information they share and less on the fact that can exchange "electrons." Since sharing information plays a defining role in infotronics technology and system design, the adoption of service-based methodologies to structure information exchange becomes a very attractive proposition. Further, with the increasing complexity of software within automobiles (Fig. 5) the need for service-based specifications of functions and features is being recognized. As a case in point, the Open Services Gateway Initiative or OSGi [10] framework has been adopted by the Automotive Multimedia Interface Collaboration or AMI-C [11].

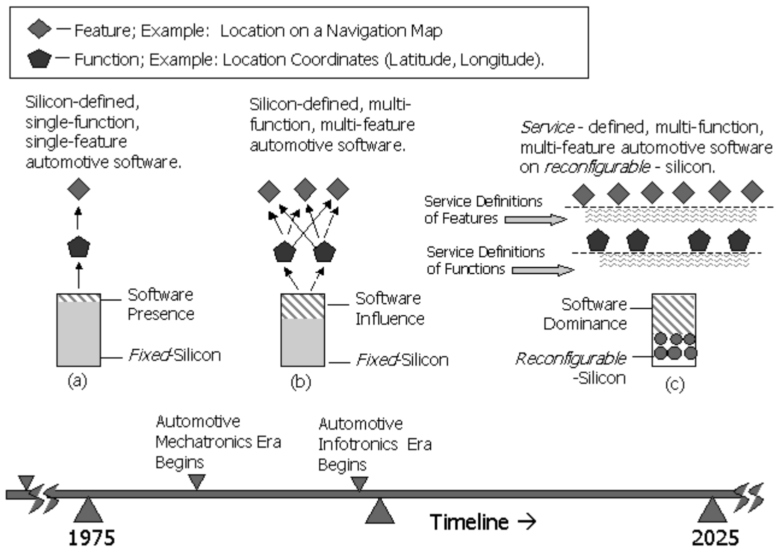

— Feature; Example: Location on a Navigation Map
— Function; Example: Location Coordinates (Latitude, Longitude).
Silicon-defined, single-function, single-feature automotive software.
Silicon-defined, multi-function, multi-feature automotive software.
Service - defined, multi-function, multi-feature automotive software on reconfigurable - silicon.
Service Definitions of Features
Service Definitions of Functions
Software Presence
Fixed-Silicon
Software Influence
Fixed-Silicon
Software Dominance
Reconfigurable -Silicon
(a)
(b)
(c)
Automotive Mechatronics Era Begins
Automotive Infotronics Era Begins
1975
Timeline →
2025

Fig. 5. Automotive Infotronics and the Evolution of Service-Defined Features

To illustrate the need for service-based definitions of infotronics systems, consider a modern luxury automobile with a navigation system and an embedded phone with a keypad built into the instrument panel. Such vehicles may also have a remote control device (much like a television set "remote") that may be used to enter destination directions into the navigation system and select map-viewing preferences all via a dedicated IrDA link. What if one stepped into such a vehicle with a personal digital assistant (PDA) that also had an IrDA transceiver? Could one "beam" a telephone number to the embedded phone via the IrDA transceiver in the navigation system? The answer today is very likely a "no." The likely reason is that the IrDA transceiver in the navigation system provides a dedicated function to the navigation system, as opposed to providing a service to the vehicle as a whole. Had the IrDA transceiver been defined by a service, say a communications service, this would have been published within an inter-networked vehicle system (Fig. 1) and if the vehicle policy (also expressed as a service) permitted it, one's PDA could have discovered the communications service and requested its use to transfer a phone number to the embedded phone. This would save the consumer the labor and frustration of having to enter a phone number, using the keypad on the instrument panel, when the consumer readily had the phone number on a PDA.

Current Practice

Current practice in automotive embedded systems is to build the application software for a module directly on the low-level infrastructure code — i.e., the RTOS or scheduler and the hardware drivers. In the simplest cases, the application code may be mixed in with the hardware drivers. The process that is used to design the module software is a traditional top down structured approach to system engineering that is sometimes referred to as the V model [12]. In this process requirements are first developed for the vehicle as a whole, and these requirements are then used do develop the requirements for the systems that make up the vehicle which in turn are cascaded to subsystems and on down to individual modules. A requirements tracking took known as CaliberRM [13] is used to perform this cascading process. In the case of our central locking example, the requirement that the module not return a "door latched" indication unless the window was also closed would be generated at the level of the subsystem that included both the door module and the central locking module.

The problem with this approach is that it does not cater for reuse of the software in modules that will be used across different vehicle lines because it does not separate generic requirements from requirements that are specific to a particular vehicle. The low-level infrastructure code can usually be reused between different vehicles, but the application code often has to be rewritten. Recently there has been an attempt to mitigate this problem by standardizing the messages that are passed between modules on a given network. Automotive message sets were initially developed on an ad hoc basis, but recently automakers have tried to manage the message sets both internally and between themselves. This is usually done by defining message sets for a specific network technology, such as CAN or MOST. Currently, these message sets are moving towards network independent domain specific protocols. For example, in the area of information and entertainment subsystems, the Automotive Multimedia Interface Collaboration (AMI-C), an organization of eight major automakers, has released a specification for a network independent Common Message Set [14].

A service based approach

The problem that arose in the central locking example is an illustration of the quandary of embedded software. It must be specific to the underlying hardware and at the same time incorporate the requirements of the overall system. As long there is a simple one to one mapping between the system and the hardware components that it contains, this presents no problem. However as soon as one wants to reuse modules across a variety of systems it becomes increasingly difficult to design module application software that will meet the requirements of all the environments in which it will be used.

In the internet domain, the corresponding dichotomy has been addressed by creating a middleware layer of software based on languages such as Java and XML. Approaches such as Jini and WSDL define this middleware layer in terms of services, providing frameworks wherein applications can discover and use services independently of the platform that they are implemented on.

This same approach has been taken in the embedded domain by the Open Services Gateway Initiative (OSGi), which has defined a framework for discovering and using services on a local area network, such as a home network, that is connected to a wide area network through an open gateway [10]. The OSGi framework uses Java to define services in a platform independent way. The OSGi specification has been used as the basis for a software platform specification by AMI-C, which has recently released a set of specifications for a vehicle services interface, a human machine interface, and off-board navigation services, as well as other core services that are required to manage software on a vehicle multimedia or telematics platform [15].

The vehicle services interface specification defines interfaces to over 80 services that may be provided by devices on the vehicle's powertrain, body and safety networks to applications running on a multimedia platform [16]. Table 1 lists some of the vehicle services that are addressed by the AMI-C specification.

Table 1. Vehicle services defined by the AMI-C vehicle service interface specification

Brake Status	Airbag Status	Engine Speed
Anti-Lock Brake Status	Convertible Top Status	Vehicle Speed
Cruise Control Status	Current Gear	Wheel Speed
Door Lock Status	Door Ajar / Trunk Open	High Resolution Odometer
Engine Running	Engine Performance Status	Vehicle Location
Exterior Lights	Engine Start Enable Status	Engine Coolant Temperature
Hazard Signal Status	Fog Lamps	Noise Level
Parking Lights	Ignition Key Status	Engine Oil Temperature
Seat Belt	Park Brake Status	Odometer Reading
Sun / Moon Roof Status	PRNDL Position	Engine Oil Level
Tire Inflation Status	Seat Occupied Status	Engine Coolant Level
Turn Signal Status	Traction Control Status	Odometer
Warning Light Status	Variable Suspension Status	Tire Pressure
Wiper Status	Window Closure Status	Vehicle Location (GPS)
System State	Power Mode Management	Vehicle Identification
Antenna Status and Control	Time / Date	Vehicle configuration
Engine Oil Pressure	Trigger Security Alert	

Like the common message set, the AMI-C software interface specifications are intended to define a set of services that are independent of platform, network and the vehicle in which they are implemented.

What is a service?

Since the specifications defined above are based strongly on the notion of services, it is appropriate to ask what is meant by a service in this context.

A service has a well-defined interface by means of which specific information can be obtained or certain actions performed. The service may consist of a single interaction between the service provider and the user of the service, or it may consist of an ongoing sequence of interactions over a specific period of time. The essential characteristic of a service is that it describes the functionality that is provided in a relatively straightforward and simple way.

Services may take several forms depending on the type of interactions that are supported. The simplest type is just a request for the service to perform some action, or to return information. For example, a vehicle audio service could return the current volume and set the volume on request. Another type of service is a subscription, where the client requests a periodic update of some information. This type of service would be used to provide vehicle speed, for example. A third type of service is one that provides notification of some event, for example, that the low fuel warning light had triggered or that the airbags had deployed. In each case, the user of the service must first make a request for the information or action that it requires. The difference is in when and how the information is returned. In some cases, more than one method would be available from the same service. For example, the door lock status service could provide both the current state of the lock for that door and a notification when the door lock status changed.

In the case of information that is updated periodically, there is a choice as to how the period is specified. The simplest case from the implementation point of view is to specify a single fixed update rate. Since vehicle signals are sampled at periods that may depend on the hardware implementation, this simplifies the service considerably. At the other extreme, the service could allow the client to specify the update rate to suit its needs. The information being returned is often not a continuously varying signal, but a stepwise constant signal whose rate of change is determined by the underlying sampling frequency of the hardware implementation of the sensor, making the latter alternative difficult to implement accurately. On the other hand, a single fixed update rate may lead to significant overhead in a process that only needs the information at a very low frequency. A common compromise is for the service to specify the minimum period, and allow the client to request updates on some fixed set of multiples of that frequency. If the base period is determined by the implementation, then the service must provide a means for the client to discover this frequency.

A related issue arises with respect to the actual content of the information returned by a service, even in response to a simple request. Different implementations of a service may provide different information about the same physical quantity. For example, some vehicles have an oil pressure sensor that returns the oil pressure as a continuous variable. Other vehicles have a simpler sensor that only provides a binary low/normal pressure reading. The question is how to specify an oil pressure service across both classes of vehicle. For most of the use cases envisioned for this service, the binary value would suffice. However, a sophisticated diagnostic application might monitor the continuously variable oil pressure to extract important prognostic information about the vehicle's powertrain. The answer could be to specify two separate services, or it could be to specify a service that can inform the client as to which type of information it provides.

The issue of matching services to requirements can be resolved by specifying properties for the service. The OSGi specification, for example allows a service to specify a set of properties when it is registered with the framework. Properties, in this context, are just name value pairs. When an application requests to be connected with a service, the service properties are used to find the best match among the versions of the service known to the framework. For this mechanism to be useful, the service providers and users must agree on a common set of property names as well as the semantics for those properties. Some service properties can be embodied in the types of the values returned by the service and the types of the inputs to the service, i.e., in its interface. However, the most interesting properties are more subtle and require domain specific definitions as in the case of the door lock status example, where one wants to know whether a returned value of “locked” implies that the windows are closed or not.

Defining an adequate set of domain specific service properties would allow one to address interesting questions, such as whether a given set of services satisfies higher-level requirements. For example, whether the set of services shown in Fig. 2 is adequate to define the lock acknowledge function or whether a window status service is also needed.

Service based architectures

In order to gain significant advantage from recasting the functionality of vehicle electronics components as services, a comprehensive framework for managing services must be defined and implemented within the vehicle domain. At the specification level this involves defining the interfaces for all of the services that will be available on the vehicle. In the automotive context, this is the responsibility of the vehicle manufacturer. The implementation of the services is then the responsibility of component suppliers. The problems that face the vehicle manufacturer are ensuring that the service specifications are consistent and that they are complete with respect to all of the technologies and features that might be included on the vehicle at a given point of time. This includes ensuring that a sufficiently rich set of service properties are defined to distinguish between versions of a service supporting all of the different levels of hardware functionality that may be present on different vehicle lines.

From the service implementer’s point of view, the main problem is ensuring that the implementation of a service or set of services conforms to the specification generated by the vehicle manufacturer. This problem requires both a sufficiently formal specification language to capture the entire service requirement and a set of tools that help with the validation of an implementation against that specification.

Conclusions

Service based software architectures have the potential to enable the creation of functionality that depends on networks of distributed embedded devices. Creating such functionality directly in application software is challenging, because the applications have to be modified whenever there is a change in the underlying hardware, or even in the configuration of the hardware. This leads to the proliferation of different versions of the application to support different hardware configurations. The creation of a middleware layer of software consisting of a set of services can help with this in a number of ways.

The advantages of a service based architecture depend on the creation of tools for the validation of service specifications, specifically tools for checking the consistency and completeness of a set of services, for validating implementations of services against their specifications and for validating network message sets against the service architecture. Addressing these needs will enable the creation of distributed electronic systems with greater functionality and reliability.

References

1. W. Gansert, T. Bertram, "Vehicle Electrical System and Circuit Diagrams," in Automotive Electrics and Electronics, 3rd Edition in English, Robert Bosch GmbH. Editor Horst Bauer, (1999) 4-5.
2. http://www.ieee.org/ organizations/history_center/milestones_photos/one_way.html
3. Y. Hirota, "Automotive Electronics and Information Technology: A Future Perspective, " JSAE, Vol. 51, No. 1, (1997) 22-28.
4. N. Allison, "Rethinking Multiplex," in Multiplexing and Networking, Editor, R. K. Jurgen, Society of Automotive Engineers, (1999) 3-11.
5. Paul Hansen “Software Tools Heading Mainstream” Hansen Report on Automotive Electronics. July/August (2003)
6. V. Bose, M. Ismert, M. Welborn, J. Guttag, "Virtual Radios," IEEE/JSAC, Special Issue on Software Radios, April (1999).
7. Michael Santarini, "ASIPs: Get ready for reconfigurable silicon," EE Times, November 20, 2000.
8. Lawrence J. Kamm, Understanding Electro-Mechanical Engineering: An Introduction to Mechatronics. Wiley-IEEE Press. (1995)
9. Vehicle Infotronics: Enabling the Integrated Mobility Experience, Proceedings of the 1998 International Congress on Transportation Electronics, Convergence 1998, Society of Automotive Engineers, October (1998).
10. Open Services Gateway Initiative: OSGi Service Platform Release 2 IOS Press Amsterdam (2002)
11. A. Malhotra, "Enabling Technology's Promise --- Collaborative Development of Common Requirements for Mobile Information and Entertainment Systems," Proceedings of the 2002 International Congress on Transportation Electronics, Convergence 2002, Society of Automotive Engineers, October (2002)
12. T. Pixton, F.Laermann, T Bietz: Vehicle Concept and Development Strategy . In The New Ford Focus, ATZ/MTZ Special Edition, Nov. (1999) 6-12
13. www.borland.com/caliber/
14. Automotive Multimedia Interface Collaboration: AMI-C Common Message Set, www.ami-c.org/publicspecrelease.asp
15. Automotive Multimedia Interface Collaboration: AMI-C Software API Specifications – Core APIs, www.ami-c.org/publicspecrelease.asp
16. Akatsuka, T., Nelson, E. AMI-C Vehicle Interface Specification. Proceedings of the IEEE Intelligent Vehicles 2001 Symposium (2001)

Modelling Electronic Service Systems Using UML*

James Skene[1], Giacomo Piccinelli[1], and Mary Stearns[2]

[1] Dept. of Computer Science, University College London, Gower Street
London WC1E 6BT, UK
{g.piccinelli, j.skene}@cs.ucl.ac.uk
[2] 2 HP Software & Solutions Operation, Pruneridge Avenue
Cupertino, CA 95014, USA
mary_stearns@hp.com

Abstract. This paper presents a profile for modelling systems of electronic services using UML. Electronic services encapsulate business services, an organisational unit focused on delivering benefit to a consumer, to enhance communication, coordination and information management. Our profile is based on a formal, workflow-oriented description of electronic services that is abstracted from particular implementation technologies. Resulting models provide the basis for a formal analysis to verify behavioural properties of services. The models can also relate services to management components, including workflow managers and Electronic Service Management Systems (ESMSs), a novel concept drawn from experience of HP Service Composer and DySCo (Dynamic Service Composer), providing the starting point for integration and implementation tasks. Their UML basis and platform-independent nature is consistent with a Model-Driven Architecture (MDA) development strategy, appropriate to the challenge of developing electronic service systems using heterogeneous technology, and incorporating legacy systems.

1 Introduction

The contribution of this paper is a UML profile for modelling systems of electronic services.

UML [16] is an object-oriented modelling language that has found broad application in analysis and design for software systems. A profile is a package of syntactic and semantic refinements for the language, which allows it to naturally model domains of interest.

An electronic service is a set of metadata, communication interfaces, software and hardware supporting a business service. A business service is a bundle of coordinated business capabilities (the content of the service) associated with provisioning mechanisms that establish the conditions under which clients, whether external or internal to the business, can access the capabilities of the service.

* This research is partly funded through the EU project TAPAS (IST-2001-34069), and the EU project EGSO (IST-2001-32409).

Business services encapsulated by electronic services benefit from additional communication and provisioning channels, but further, they permit the automated coordination of capabilities, resources and information, both within and between organisations. This gives rise to Electronic Service Systems (ESSs), in which the services are integrated using auxiliary components such as workflow engines for coordination, databases to store knowledge about the state of the enterprise, and Electronic Service Management Systems (ESMS),which we characterise in this paper as combining these various capabilities to provide viewpoints and control of the enterprise to management, citing experience of the HP Service Composer and the DySCo (Dynamic Service Composer) research prototype.

The challenge on the business side is to adapt business infrastructure and models to service-oriented principles. The challenge on the technical side is to provide integration solutions that are accessible, comprehensive and beneficial. This requires thorough understanding and active management of the relationships between business capabilities and technical infrastructure. Such understanding and management can be achieved through modelling. This modelling serves as a starting point for software implementation, integration and provisioning tasks, which must be applicable to electronic services realised using a variety of technologies. The Model Driven Architecture (MDA) is a software development strategy based on UML models that explicitly addresses the challenge of integration of heterogeneous systems, and we therefore choose UML as a basis for our modelling, to ensure compatibility with this approach.

Determining strategies for coordinating services can be difficult, due to complex dependencies between services and the large number of possible states for the enterprise, arising from the parallel evolution of multiple services. We associate a formal model of behaviour based on workflows with our models of services, providing the opportunity for analysis. This model also formalises our notions of coordinated capabilities forming larger conceptual entities such as services, and the flows of information resulting from service enactment. Finally, the formal semantic provides a reference for implementation activities proceeding from our models, giving developers the opportunity to assert that software components act as required.

The remainder of this paper is structured as follows: In Section 2 we provide a background with a discussion of electronic services (Section 2.1), and UML and the MDA (Section 2.2); in Section 3 we present a meta-model describing the domain of ESSs; in Section 4 we show the translation of the meta-model into a UML profile; in Section 5 we discuss related work and then summarise in Section 6.

2 Background

2.1 Electronic services

The notion of a 'business service' enables the management within an enterprise of 'capabilities' to deliver some benefit to a consumer. The term 'capability' refers to the coordination of simpler tasks to achieve an end; the concept is used to raise the level of abstraction when describing the way that a business behaves. When describing business services, capabilities are divided into those involved in 'provisioning' the service, and those providing the 'content' of the service. The content of a service is the set of capabilities that deliver the benefit of the service to the client. For example, the content of a freight service refers to the capability of moving goods from one place to the other. Provisioning refers to the business channel [6] between the provider and the consumer of a service. In the example, provision covers selection, product offer, pricing, and interaction processes that the freight company applies to its customers. Content and provisioning are complementary aspects of a service. On the one side, the provisioning logic depends on the capabilities that the provider can support. On the other side, the capabilities made available to consumers depend on the provisioning logic adopted by the provider. In the example, the option of delivery tracking might be made available only to selected customers. The example is based on previous research in the freight domain [12].

An electronic service is a business service with communication and coordination aspects implemented using computer systems [14].

Because business services require communication between the provider and the consumer it is natural to provide interfaces to business services using communication technologies such as computer networks, and the software that supports this such as middleware for distributed systems. Indeed, a service metaphor is widely used in these technologies. Listeners on network interfaces are often referred to as services, and web-services communicate using Internet protocols to provide services of all sorts. Such services are closely analogous to business services, even to the extent of exhibiting a separation between provisioning capabilities in the form of meta-data interfaces, reflection and directories, and the back-end logic implementing the content of the service. Web-services conform to the model further, by including business terms in meta-data [24], enabling a market in services.

Middleware services and computing resources also provide the opportunity to implement new business services with a highly automated content, and this is an expected benefit of the electronic service model. However, despite the similarities, our notion of electronic services should not be confused with middleware services. Services must also be coordinated: Internally, to marshal the involved capabilities and resources and establish the relationship between content and provisioning; and externally, to manage the interaction between the service and its clients and environment. This coordination requires a view of the behaviour of a service. We therefore introduce an operational semantic for capabilities, presented in Section 3.2. This semantic is broadly compatible with workflow languages, suggesting that services could be both coordinated and enacted by workflow engines.

Our semantic also describes abstractly the effect that activities have on the information in their environment, for example the known locations of vehicles, or statistics such as the total revenue for a service. Such information can have a role in coordinating capabilities, and may be maintained and leveraged using databases or other accounting mechanisms.

There is also a need to manage the resources required by a service, which may be the role of an Enterprise Resource Planning (ERP) application. Generally, if electronic services are in place there will be the need and opportunity to integrate them using a technical infrastructure. We introduce the notion of an Electronic Service Management System (ESMS), informally defined as an application that includes coordination, information and resource management capabilities, providing business-oriented viewpoints and control over the services that it manages.

IT technology trends and the service model for business provide the context for electronic services. An enterprise adopting an electronic service strategy would structure its business as services, provide interfaces to those services using middleware technologies, coordinate and automate the services from a workflow-oriented perspective and implement a technological infrastructure to take advantage of the coordination and communication opportunities that are the key benefit of the electronic service model.

2.2 UML and the model-driven architecture

Businesses adopting an electronic service strategy will be faced with integration and implementation challenges. Modelling systems of electronic services is a vital step towards meeting a number of these. When implementing a new electronic service, or adapting an existing business service to an electronic service it is necessary to understand the intended environment of the service, and its interaction with other services and management systems. Similarly, when introducing new management components, it is necessary to have a clear understanding of the services with which it will interact.

The Unified Modelling Language (UML) is an object-oriented graphical language that has found wide applicability in analysis and design for software engineering. In this paper we provide a profile for UML to allow the modelling of ESSs. Profiles are an extension mechanisms whereby the innate notations provided by the UML can be augmented with labels, called 'stereotypes', tagged values and constraints, which provide semantic refinement, annotations and syntactic refinement respectively.

UML is based on a conceptual architecture that is divided into four meta-modelling layers as shown in Figure 1. The lowest level is the data layer (M0), in which objects such as data-patterns in computer memory and other real-world phenomena including people and things are supposed to reside. The elements in the lowest level are classified by types in the UML models that analysts and designers produce, which hence reside at the next meta-level (M1). UML model elements are, in turn, objects of classes in the UML meta-model (M2). Attached to these meta-classes are semantic descriptions and syntactic constraints that control the meaning and applicability of the UML. The meta-model at level M2 is self-describing, so can also be regarded as residing in level M3 (and plausibly all higher levels).

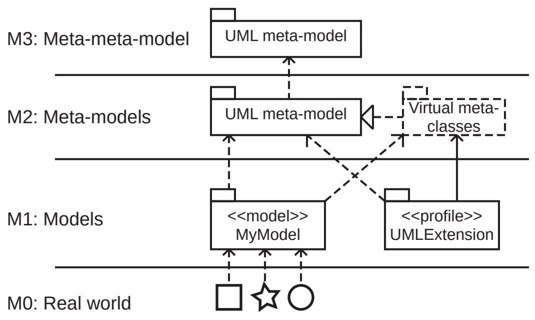


Fig. 1. Meta-modelling architecture of the UML

Profiles then, are a means of refining classes, semantics and syntactic constraints at the M2 level. Confusingly, profiles exist at the M1 level, so that they can be denoted using UML and deployed by including them with any UML model that requires their language extensions. They can therefore be regarded as injecting 'virtual meta-classes' into the UML meta-model (M2).

Before presenting our profile, we present a meta-model that is similar to the UML meta-model, and can be considered to reside at level M2 in the conceptual architecture. This is a common practice when defining profiles [5], as a new meta-model describes the semantic domain directly, independently of the need to refine the semantics of the UML meta-model. In section 4 the meta-model is mapped onto profile elements, and existing elements in the UML meta-model. Our meta-model therefore serves as both a reference model for our definition of electronic services and to define the semantics of the profile.

The Model Driven Architecture (MDA) [17] is a modelling approach based on UML. It recommends that development organisations separate models of their business logic (Platform Independent Models - PIMs), from technical artifacts, such as design models (Platform Specific Models - PSMs) and source code. The benefit is to insulate organisations from the cost of re-deploying software services as architectural infrastructures change, particularly middleware standards. The approach also supports the integration of heterogeneous and legacy software, and for these reasons is extremely well suited to development tasks in an electronic service environment. In the terminology of the MDA, the models produced using our profile are Platform Independent Models (PIMs). UML can represent refinement relationships between models, for example between a PIM and a PSM. Our models can therefore be related to more detailed design models, supporting the MDA approach when implementing or integrating electronic services.

In supporting an MDA approach our profile is similar and complementary to other profiles including the standard Enterprise Distributed Object Computing (EDOC) profile [18], which can be used to represent enterprise computing systems in a platform-neutral manner.

3 The ESS meta-model

The ESS meta-model is divided into two packages as shown in Figure 2. These partition the elements pertaining to services from those which represent management applications. The management component metamodel naturally depends on concepts from the service metamodel. The following sections present these metamodels in detail.

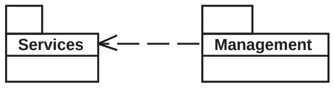


Fig. 2. Subpackages within the ESS meta-model

3.1 The service meta-model

Figure 3 shows the part of the services meta-model related to the composition of capabilities into services. The elements shown are now described:

Service An electronic service as described in Section 2.1. Services have any number of provisioning capabilities, and a single top-level content capability (the capability to deliver the service). Services can be composed of sub-services, in which case the content capability coordinates the content of each sub-service, and each sub-service must have a provisioning capability that makes a service offer to a role in the coordinating content capability.

Capability A business capability described by a workflow. The behaviour of capabilities is described formally in Section 3.2. Informally, a number of roles perform actions and cooperate to complete some task. Capabilities can be composed in a hierarchy. The workflow of a coordinating capability constrains the order of tasks in the component capabilities.

CapabilityRole A capability role identifies the behaviour of a worker or resource in a coordinated task. Capability roles can be assigned to actual business entities as discussed below.

InformationItem An identifier for a piece of information about an enterprise that is relevant to a task. Some workflow actions require information as a prerequisite and produce or process information as by-product of their enactment.

Observation Observations give rise to new information from existing information. This captures the idea that not all derived information is produced by a particular action. When the condition of the observation is satisfied then new information may be introduced by the observation expression.

Constraints defined over the meta-model further reinforce these informal semantics. For example, capabilities may not coordinate themselves. Constraints are expressed formally using OCL [16]:

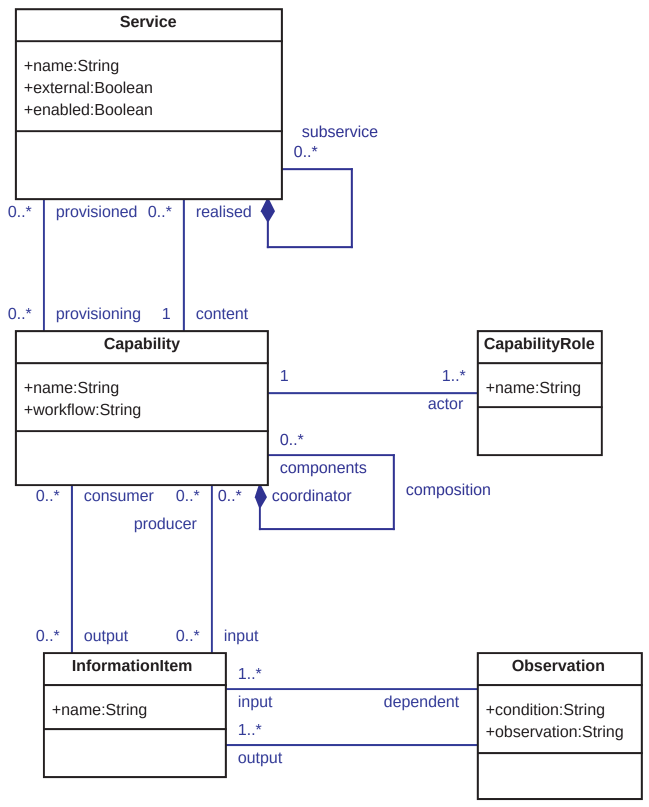


Fig. 3. Capabilities view of the services meta-model

```
context Capability
  def:
    let allCoordinators = self.coordinator→union(
      self.coordinator→collect(c | c.allCoordinators))
  inv:
    not self.allCoordinators→exists(c | c = self)
```

Complementary to the abstract view of services are models of the business assets in an enterprise, and their assignment to capability roles to realise a service. Figure 4 shows the meta-model classes supporting such models.

BusinessEntity A business entity is a person, resource or system that can fulfil one or more roles in a capability.

ServiceOffer A service offer is made to a capability role (typically that of the ‘customer’). That capability role must be associated with one of the provisioning mechanisms of the service.

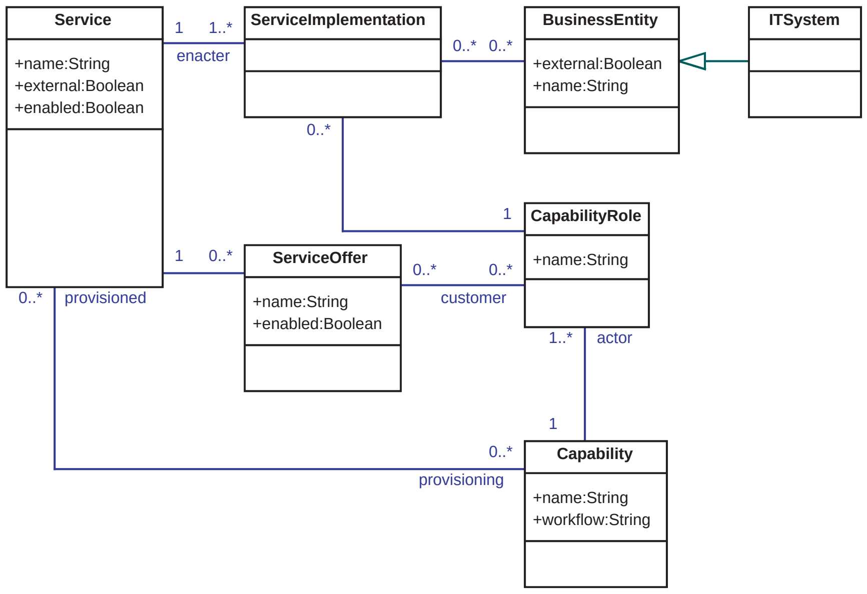


Fig. 4. Implementation view of the services meta-model

ServiceImplementation ServiceImplementation captures the idea that business assets can be assigned to capability roles in order to make a service concrete. There is no explicit notion of service instance. However, if necessary business assets can be grouped to show those relevant to a particular scenario.

ITSystem An IT system is a computing system that can perform a role in a capability. Electronic services are intended to provide integration and automated coordination. This class allows the identification of the components providing these services, possibly as a prelude to an MDA-style development activity. Section 3.3 provides refinements of this stereotype to identify likely management applications.

Additional classes not shown in Figures 3 and 4 are now discussed:

Property and HasProperties Properties capture different types of meta-data about capabilities. Such meta-information mainly refers to functional and non-functional requirements for a capability. For example, a property for a negotiation capability is to be usable only with a certain type of customers. The following classes inherit from HasProperties to enable the attachment of properties: BusinessEntity, CapabilityRole, Capability and Service. The properties mechanism maps onto the tagged-value mechanism in UML in the profile definition.

Group and Groupable Experience with the HP Service Composer revealed the benefit of composing capabilities into loosely-grouped higher-level aggregates called ‘clusters’, in which capabilities exhibited functional overlaps, dependencies, mutual ownership or other subjective similarities. There is also often the need to group services into related offerings or ‘service packs’. Finally, as stated above, a grouping mechanism addresses the lack of a concept of service instance by allowing the association of business entities that actually cooperate (since more than one entity can enact a given service role). Group and Groupable provide a single mechanism for hierarchical grouping. The following elements inherit from Groupable, and hence may appear in a Group: CapabilityRole, Capability, BusinessEntity, InformationItem, Service and Group. Grouping is implemented by UML’s package mechanism in the profile definition.

3.2 Formal semantics for the service meta-model

In this section we formalise notions of information and coordination for capabilities, using the Structured Operational Semantics (SOS) style of [22], in which inference rules define the structure of a Labelled Transition System (LTS) intentionally. This definition contributes to the semantics of the profile by emphasising the definition of capabilities as coordinated activities whose behaviour is known, and by providing a high level constraint on workflow descriptions taken as values for the meta-attribute Capability.workflow. Our formalism is defined independently of specific workflow languages by omitting base cases for our rules. Instead, we assume that the workflow language employed allows us to make assertions such as:

$$\langle \Sigma \cup I, c \rangle \xrightarrow{\alpha : I \to O} \langle \Sigma \cup O, c' \rangle \tag{1}$$

Meaning that a specific, isolated capability, c, in a system where the current information is represented by $\Sigma \cup I$, evolves to c' by undertaking an action, α, which effects some change, reflected by the transformation of the information I to new information O.

Capabilities may evolve independently of each other, when not coordinated:

$$\frac{\langle \Sigma, c_i \rangle \xrightarrow{\alpha} \langle \Sigma', c_i' \rangle}{\langle \Sigma, \{c_1 \dots c_i \dots c_k\} \rangle \xrightarrow{\alpha} \langle \Sigma', \{c_1 \dots c_i' \dots c_k\} \rangle} \tag{2}$$

Even when coordinated, capabilities may perform uncoordinated actions ($A(c)$ yields the set of actions that a process c can undertake):

$$\frac{\langle \Sigma, c_i \rangle \xrightarrow{\alpha} \langle \Sigma', c_i' \rangle \qquad \alpha \notin A(c_c)}{\langle \Sigma, c_c[\{c_1 \dots c_i \dots c_k\}] \rangle \xrightarrow{\alpha} \langle \Sigma', c_c[\{c_1 \dots c_i' \dots c_k\}] \rangle} \tag{3}$$

Coordinated actions may occur only when the coordinating process permits, and when all capabilities that can perform them are ready to do so simultaneously:

$$\frac{\langle \Sigma, c_c\rangle \xrightarrow{\alpha} \langle \Sigma', c_c'\rangle \quad \langle \Sigma, c_1\rangle \xrightarrow{\alpha} \langle \Sigma', c_1'\rangle \ldots \langle \Sigma, c_i\rangle \xrightarrow{\alpha} \langle \Sigma', c_i'\rangle \quad \alpha \notin \bigcup_{c_f \in F} A(c_f)}{\langle \Sigma, c_c[\{c_1 \ldots c_i\} \cup F]\rangle \xrightarrow{\alpha} \langle \Sigma', c_c'[\{c_1' \ldots c_i'\} \cup F]\rangle} \tag{4}$$

A capability may have multiple coordinators in the metamodel. The interpretation of this is that the capability is a subcapability of its coordinator. It is therefore replicated for each coordinator. Shared sub-capabilities are not synchronized.

Note that an action may require certain information to be present and satisfy some condition before the action can be performed. Hence, coordination by shared memory is also possible for capabilities. Under the electronic service model, provisioning and content capabilities are not explicitly coordinated, hence this mechanism links these capabilities for a service. The provisioning capabilities create conditions under which the content capabilities are enabled.

Information in the system may arise naturally from the occurrence of actions. However, the progress of the system may depend on broader observations than those made in the context of a particular action. Hence we enable the modelling of observations that derive new information from that already present in the system:

$$\frac{\langle \Sigma \cup I, \Gamma\rangle \wedge \exists o : I \rightarrow O \in \Omega}{\langle \Sigma \cup I \cup O, \Gamma\rangle} \tag{5}$$

We do not prescribe the language used to specify observations. OCL would be a good candidate. The information prerequisites for the observation could be captured by a boolean expression, and then **let**-clauses could introduce new information. Note that it is possible to specify observations that lead to inconsistencies in the system information. Modellers should try to avoid this. One strategy for dealing with this is to rule that if multiple values can be derived for an information item then the value of the information item is not known. However, in systems where action is preferable to inaction, this may not be safe.

For the purposes of assigning work the underlying workflow language must also associate actions with roles, although this association is not required in this discussion of coordination, as we assume that coordination is independent of the entities that implement roles. That is, an entity will eventually be capable of enacting all actions required of it during the evolution of the system.

The benefit of a formal semantic based on an LTS are in terms of simulation and analysis. A tool such as LTSA [13] can provide scenario-based validation of models. This can be used to assert safety conditions, fairness and liveness conditions, and to ensure the absence of deadlocks (presumably arising from capabilities failing to establish adequate preconditions for their successors). The use of information for coordination complicates such models, and can increase their state-space beyond feasibility. However, reasonable abstractions can usually be found.

3.3 The management meta-model

The management meta-model shown in Figure 5 allows the identification of common management components and their relationship to electronic services. We have not included modelling functional or structural relationships between management components as this is out of scope of our discussion of electronic services. However, such modelling is necessary and is supported by the full expressive power of the UML, possibly augmented by other profiles such as the EDOC profile.

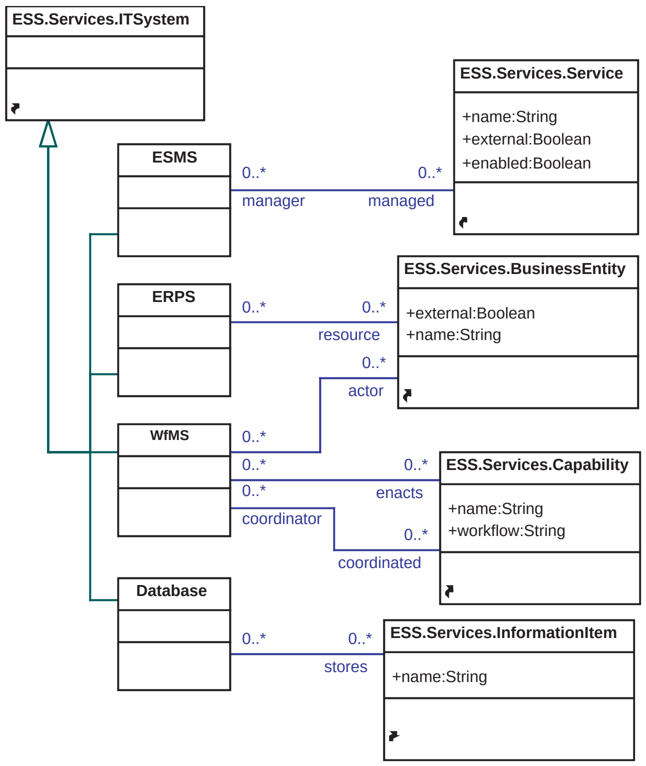


Fig. 5. The management meta-model

ESMS An application offering a enterprise-oriented management view of an electronic service environment. For example, the HP Service Composer [7], or the DySCo research prototypes [21]. Other candidate technologies might be an application service offering a middle-tier of business logic, with a web-server providing the management interfaces.

WfMS A workflow management system, either embodying a capability (enactment) or coordinating a number of subcapabilities. Examples of workflow applications are IBM's MQ-Series Workflow [8] and PeopleSoft's [20] PeopleTools and Integration Broker.

ERPS An Enterprise Resource Planning System, dedicating to coordinating entities in the system, presumably making them available to fulfil capability roles. We do not consider resource planning in this paper, although it interacts at a functional level with coordination based on capabilities, and future work may provide a combined modelling approach. Examples of ERP systems are SAP's mySAP [23] and Baan's iBaan [1].

Database Most enterprises use databases to store information about the enterprise. Establishing a relationship between the (conceptual) information items and the databases that store them allows a modeller to check whether the information required by a business entity to fulfil a capability role is available in its context. Popular databases are Oracle [19] and MySQL [15].

4 The ESS Profile

The following tables relate elements in the meta-model to profile elements and elements in the UML meta-model.

All name attributes in the meta-models map to the name attribute of the class element in the UML meta-model. All associations in the meta-model map to associations in models. Stereotypes on AssociationEnds are used to disambiguate associations where more than one exists between the same two meta-model elements. The meta-model constraints also have translations into constraints on the profile elements, and additional constraints reflect the structure of the original meta-model. For example, the 'Fulfills' stereotype can only be attached to an association between a CapabilityRole and a BusinessEntity, and its service tag must always be present:

```
package Foundation::Core
  context Association
    inv:
      self.stereotype→exists("Fulfills") implies
        self.connection.participant.stereotype→exists("CapabilityRole")
        and
        self.connection.participant.stereotype→exists("BusinessEntity")
        and
        self.taggedValue.type→exists(name = "service")
```

5 Related work

The definition and characteristics of the ESS model derive substantially from the experience of HP Service Composer. UML notation is used in the HPSC, with a separation between platform-dependent and platform-independent models of an electronic service. Workflow notation and technology is used to model and manage the business logic of a service.

Meta-model element	Stereotype	UML base class	Parent	Tags
Service	Service	Class	–	external enabled
Service.content	content	AssociationEnd	–	–
Service.provisioning	provisioning	AssociationEnd	–	–
Service.component	component	AssociationEnd	–	–
Capability	Capability	Class	–	
Capability.input	input	AssociationEnd	–	–
Capability.output	output	AssociationEnd	–	–
CapabilityRole	CapabilityRole	Class	–	–
InformationItem	InformationItem	Class	–	–
Observation	Observation	Class	–	condition observation
Observation.input	input	AssociationEnd	–	–
Observation.output	output	AssociationEnd	–	–
BusinessEntity	BusinessEntity	Class	–	–
ServiceOffer	ServiceOffer	Class	–	enabled
ServiceImplementation	Fulfills	Association	–	service
ITSystem	ITSystem	Class	BusinessEntity	–
ESMS	ESMS	Class	ITSystem	–
WfMS	WfMS	Class	ITSystem	–
WfMS.actor	wfactor	AssociationEnd	–	–
WfMS.enacts	enacts	Class	–	–
WfMS.coordinated	coordinates	Class	–	–
ERPS	ERPS	Class	ITSystem	–
Database	Database	Class	ITSystem	–

Table 1. Stereotypes in the ESS profile

Meta-model element	Tag	Stereotype	Type	Multiplicity
Service.external	external	Service	Boolean	0..1
Service.enabled	enabled	Service	Boolean	0..1
Capability.workflow	workflow	Capability	String	0..1
Observation.condition	condition	Observation	String	1
Observation.observation	observation	Observation	String	1
ServiceImplementation.service	service	Fulfills	Class	1
BusinessEntity.external	external	BusinessEntity	Boolean	0..1
ServiceOffer.enabled	enabled	ServiceOffefr	Boolean	0..1

Table 2. Tags in the ESS profile

The ESS model is also closely related to the DySCo (Dynamic Service Composer) [21] research prototype. DySCo is the result of a two-year project involving University College London (UK), the University of St. Petersburg (Russia), the University of Ferrara (Italy), the University of Hamburg (Germany), and

Hewlett-Packard (UK and USA). The objective of DySCo was the development of a conceptual and technology framework for the dynamic composition of electronic services. While lacking direct support for UML, DySCo provides modelling facilities for workflows and a homogeneous execution platform for an ESMS.

An electronic-services model is currently being used in the context of the EGSO (European Grid for Solar Observations) [2] project. The model-driven approach to the architecture of the service provision part of the EGSO grid is expected to address the need to integrate services based on different provision models and execution platforms. Each service provider in the EGSO grid will be equipped with an ESMS. In addition, a specific ESMS federates and manages the service provisioning capabilities of the overall EGSO grid.

The Enterprise Collaboration Architecture (ECA) defined in the OMGs EDOC specification [18] provides a comprehensive framework for the modelling of enterprise systems. The ESS profile introduces enterprise system components that can be designed based on the ECA, and provides a means to model features peculiar to electronic services that are not explicitly addressed by the ECA. Similar considerations apply for the Reference Model for Open Distributed Processing (RM-ODP) [9], which is also closely related with the ECA.

Most technology and conceptual frameworks for electronic services [11] focus on web-service-based automation of the front-end of individual services. Web Services [3, 4] constitute the reference model for access to and basic orchestration of business resources. We envision Web Services playing a fundamental role in the realisation of electronic services. Still, a more comprehensive approach is needed for the realisation and operation of business-level services. An example of the issues involved in the realisation of business-level service is HiServs Business Port [10]. FRESCO (Foundational Research on Service Composition) [22] provides an example of second-generation framework for electronic service management. The focus of FRESCO is on the provision aspects of services.

6 Conclusions

Electronic services provide the conceptual and technology framework for the aggregation and coordination of business resources. The realisation and operation of a service requires close integration between different systems. A model-driven approach to development in an electronic-service environment helps tackle the integration issues arising from heterogeneity and change.

In this paper, we present a means to model ESSs using UML, in a manner compatible with the MDA approach. We apply concepts derived from the specific experience of HP Service Composer, but also closely related to concepts in OMGs EDOC specification and the RM-ODP. The semantics of the models are described with reference to a meta-model from which a UML profile is defined. The behaviour of electronic services is described formally using operational semantics, providing an additional benefit of our models as a foundation for formal analyses.

References

1. Baan. *iBaan*. http://www.baan.com/.
2. R. D. Bentley. EGSO – the european grid of solar observations. In *European Solar Physics Meeting, ESA Publication SP-506*, 2002.
3. E. Cerami. *Web Services Essentials*. ORielly and Associates, 2002.
4. M. Clark et Al. *Web Services Business Strategies and Architectures*. Expert Press, 2002.
5. D. S. Frankel. *Model Driven Architecture: Applying MDA to Enterprise Computing*. John Wiley and Sons, 2003.
6. B. Gibb and S. Damodaran. *ebXML: Concepts and Application*. John Wiley and Sons, 2002.
7. Hewlett-Packard Company. *HP Service Composer User Guide*, 2002.
8. IBM. *Websphere MQ Workflow*. http://www-3.ibm.com/ software/ integration/ wmqwf/.
9. ISO/IEC, ITU-T. *Open Distributed Processing – Reference Model – Part 2: Foundations, ISO/IEC 10746-2, ITU-T Recommendation X.902*.
10. R. Klueber and N. Kaltenmorgen. eServices to integrate ebusiness with ERP systems – the case of HiServs business port. In *Workshop on Infrastructures for Dynamic Business-to-Business Service Outsourcing (CAISE-ISDO)*, 2000.
11. H. Kuno. Surveying the e-services technical landscape. In *Workshop on Advance Issues of E-Commerce and Web-Based Information Systems (WECWIS)*. IEEE Press, 2000.
12. N. Linketscher and M. Child. Trust issues and user reactions to e-services and e-marketplaces: A customer survey. In *DEXA Workshop on e-Negotiation*, 2001.
13. J. Magee and J. Kramer. *Concurrency: State Models and Java Programs*. John Wiley and Sons, 1999.
14. A. Marton, G. Piccinelli, and C. Turfin. Service provision and composition in virtual business communities. In *IEEE-IRDS Workshop on Electronic Commerce*, Lausanne, Switzerland, 1999.
15. MySQL AB. *MySQL Database*. http://www.mysql.com/.
16. OMG document formal/2003-03-01. *Unified Modelling Language (UML), version 1.5*, January 2003.
17. OMG Document ormsc/01-07-01. *Model Driven Architecture (MDA)*, July 2001.
18. OMG, document ptc/02-02-05. *UML Profile for Enterprise Distributed Object Computing Specification*, May 2002.
19. Oracle. *Oracle database products*. http://www.oracle.com.
20. PeopleSoft. *PeopleTools and Integration Broker*. http://www.peoplesoft.com/.
21. G. Piccinelli and L. Mokrushin. Dynamic e-service composition in DySCo. In *Workshop on Distributed Dynamic Multiservice Architecture, IEEE ICDCS-21*, Phoenix, Arizona, USA, 2001.
22. G. Piccinelli, C. Zirpins, and W. Lamersdorf. The FRESCO framework: An overview. In *Symposium on Applications and the Internet (SAINT), IEEE-IPSJ*, 2003.
23. SAP. *mySAP*. http://www.sap.com/.
24. UDDI.org. *UDDI (Universal Description, Descovery and Integration) Executive White Paper*, November 2003. http://www.uddi.org/ pubs/ UDDI_Executive_White_Paper.pdf.

User Quality of Perception: Towards a Model for Personalised Communication Services

Cláudia Maria F.A. Ribeiro[1,2], Nelson Souto Rosa[1], Paulo Roberto F.Cunha[1]

[1] Universidade Federal de Pernambuco
Cidade Universitária, Recife-PE, Brazil
[2] Universidade do Estado do Rio Grande do Norte
Campus Universitário, Mossoró-RN, Brazil
{cmfar,nsr,prfc}@cin.ufpe.br

Abstract. The emerging of the new generation applications like videoconference and the increasing in the demand of QoS services has enforced the need of new service models. As in the most new applications the user can perceive the level of quality a service is provided, data communication services are currently evolving towards more personalised ones. A direct consequence of this trend is the necessity of explicit treatment of the user quality of perception (known QoP). Challenges in this evolution include the better understanding of "how" users perceive QoS and "how" the perception is actually realised by underlying QoS mechanisms. We present a formal architecture, namely ESCHER that implements a QoP model for QoS services. The QoP model focus the precise specification and mapping of user QoS requirements into QoS parameters. The Z notation is used to formalise the ESCHER elements and their operations

1. Introduction

The emerging of new generation applications like videoconference and the increasing in the demand of QoS services have motivated some key transformations in the application development process. Meanwhile, there is a clear necessity of new service models [1].

Data communication services are currently evolving towards more personalised ones, as the users can perceives the level of quality a service is provided [4]. In fact, those services are becoming as personalised as health care services, bank services and traditional voice communication services [2]. A direct consequence of this trend is the necessity of explicit treatment of the user quality of perception (known QoP).

Typical users are not able to express their QoS requirements in quantitative terms, as they are not concerned with details of implementation of QoS services. For instance, they know neither what is the upper limit of tolerable packet *delay* nor *jitter* in an IP telephony session. Moreover, they cannot provide the traffic specification of their application flow.

Actually, the user has a very subjective view of QoS and he/she usually defines QoS constraints as a set of non-functional requirements (NFRs) such as performance and cost. In order to understand, precisely specify and map user QoS specifications into quantitative network parameters, new capabilities must be incorporated by QoS mechanisms.

Despite the high abstraction level in which NFRs are commonly stated, there is a rationale to treat with the NFRs defined by users. As resources are traditionally scarce, the resource allocation based on the quality perceived by the user yields a more effective resource management. For example, a video quality may be good for a particular user, while its quality is not acceptable to others. The optimisation of resource allocation embodies benefits to communication service providers, whilst the differentiation of services motivated by the user quality of perception (QoP) leads to money saving on behalf of the users.

Quality of Service is a key factor for differentiating service offers in a competitive market. Some organisations of communication service providers such as Eurescom and ETSI are currently trying to define service quality classes (QC model), which are easily identified by users [5,12]. For instance, the class of communication service named "gold" would be associated to a better quality of service and consequently with a higher cost than the class "silver".

The aforementioned initiative are a progress towards the effective treatment of the individual necessities of users, however two important issues are still open: it is necessary a better understanding of "how" the user perceives quality; and "how" the perception is actually realised into underlying quality of services elements.

We have addressed these mentioned issues by conceiving a layered architecture namely ESCHER[1] that implements the QoP model for QoS services. The QoP model focuses on both the precise specification and mapping of user QoS requirements into QoS parameters. The precise specification and its benefits are obtained by the adoption of the Z notation.

The definition of the ESCHER architecture has four main benefits. Firstly, it contains a layer that enables the active participation of users in the definition of QoS requirements, preserving their subjective notion of QoS. Second, the architecture makes explicit the elements involved in the specification and provisioning of personalised QoS services in different levels. Thirdly, it serves to guide the implementation of personalised QoS services. Finally, it facilitates the mapping of high abstraction level specification (defined by the user) into more concrete specification (treated by underlying QoS mechanisms) through the integration of different QoS views.

This paper is organised as follows. Section 2 introduces the QC model. Section 3 presents the principle of the QoP model. Sections 4 and 5 are dedicated to ESCHER QoP architecture fundamentals and formalisation, respectively. In Section 6 is presented a modelling of videoconference application using ESCHER. Finally, the last section presents the conclusions and some directions for future work.

[1] Maurits Cornelis Escher (1898-1972) is a graphic artist. He is most famous for his so-called "impossible" structures which allow multiple perceptions

2. Related Work

There are some research activities on capturing of user QoS requirements [10,11,13]. However, due to space limitation and mainly for relevance to our proposal, only a Quality Class Model (QC Model) named QUASI-model [5] is described in this section.

In a typical scenario of QUASI-model, the user requires a certain Quality Class (QC) for a specific Application Category (AC) to the Service Provider. Each combination (QC, AC) corresponds to a Network Performance Level (NPL).

Some examples of Application Categories include: 'Conversational', 'Streaming', 'Interactive' and 'Background'. The QUASI-model also defines the Quality Classes (QCs) that are used as a QoS differentiator towards the user. Examples of Quality Classes are 'Gold' and 'Silver'.

An NPL is a small set of network performance parameters, e.g., delay, jitter and loss. Each parameter has assigned a value range and the guarantee that each parameter value belongs is in the range. This represents the goal of the network in order to provide the desired Quality Class of a particular Application Category.

The QC model assumes that the user has selected a QoS (offered by the network provider) based on his/her perception. This fact implies that the network provider or the application service provider offers a limited set of QoS profiles that the user can chose. The selected QoS is mapped into attributes applicable to the QoS mechanism that realises the requested QoS. When the QoS mechanism has accomplished the selected QoS, the transmitter can send the content to the user.

Table 1. QC model example

	AC1 Non-Real Time	**AC2** Non-Interactive Real Time	**AC3** Interactive Real Time
Premium	NPL_{11}	NPL_{12}	NPL_{13}
Basic	NPL_{21}	NPL_{22}	NPL_{23}

A practical QC Model (see Table 1) contains two Quality Classes, namely *Premium* and *Basic*, and three Application Categories, namely *interactive non-real-time*, *non-interactive real time* and *interactive real-time*. Combining the two QCs with the three ACs results at most in six possible data flows to be treated differently in the network in order to preserve the performance requirements of the relative services/applications.

Despite the simplicity, the main disadvantage of QC Model refers to the categorisation of users. The different level of perception of each user is not direct treated by this model. Additionally, the cost is a consequence of selected quality class and NPL associated, not a constraint imposed by the user.

The QoP Model, described in next Section, presents a service model to solve the questions above mentioned, by considering the explicit treatment of user quality of perception. It serves as a basis of QoS management service implemented by ESCHER architecture proposal.

3. The QoP Model

The QoP Model considers the necessity of precisely expressing QoS requirements at user level, the necessity of mapping QoS requirements into QoS parameters of underlying QoS mechanisms and the need of guaranteeing the required QoS level. Some basic principles are considered by QoP model:

- The user can require different level of quality for multimedia services;
- The QoS provided is perceived by the user by considering personal characteristics;
- The user satisfaction level is represented by the trade-off between non-functional requirements like QoS and cost;
- The QoS mapping process translates subjective specification defined by the user into QoS parameters used by underlying QoS mechanisms;
- Each QoS mechanism has its own set of QoS parameters; and
- The QoS mapping is a bi-directional process: from upper to lower layers at configuration phase and in the opposite direction at monitoring phase.

To realise aforementioned principles, in QoP Model the “user layer” is proposed as the most abstract layer instead the traditional “application layer”. In traditional approach, the needs of the users are dealt in an implicit manner by the application. In a personalised service, the user has the opportunity of defining the level of desired quality and the upper limit for the cost. The perception of QoS is a function of personal characteristics of each user.

Another difference between traditional approach and QoP Model is the way the user satisfaction level is considered. In QoP Model the user satisfaction level is represented by the trade-off between non-functional requirements like QoS and cost.

A key feature of QoP Model is the separation of concerns in the requirement specification. By adopting this principle, both the specification of non-functional requirements (including QoS requirements) and functional requirements of a given application, can be obtained in a separate manner. It allows legacy applications to use QoS services without any change in its code. In the proposed QoP architecture described in Section 4, the middleware layer that serves as an architectural element for integrating requirement specifications [8,9] has this characteristic.

It is important to notice that each QoS mechanism has its own set of QoS parameters and the QoS mapping has to consider this aspect [7]. In QoP Model the QoS mechanism are represented in an abstract way by an architectural layer. This feature allows the introduction of new mechanisms.

Finally, the QoP Model considers two phases for QoS management service: configuration and monitoring. The first one includes QoS requirements specification, QoS mapping and QoS negotiation for resource allocation. The second one represents the renegotiation phase when environmental changes affect the level of QoS provided. The QoS mapping is structured as bi-directional process: from upper to lower layers at configuration phase and in the opposite direction at monitoring phase.

4. The ESCHER QoP Architecture

Figure 1 illustrates the ESCHER layers and the relationships between QoS abstractions of each layer.

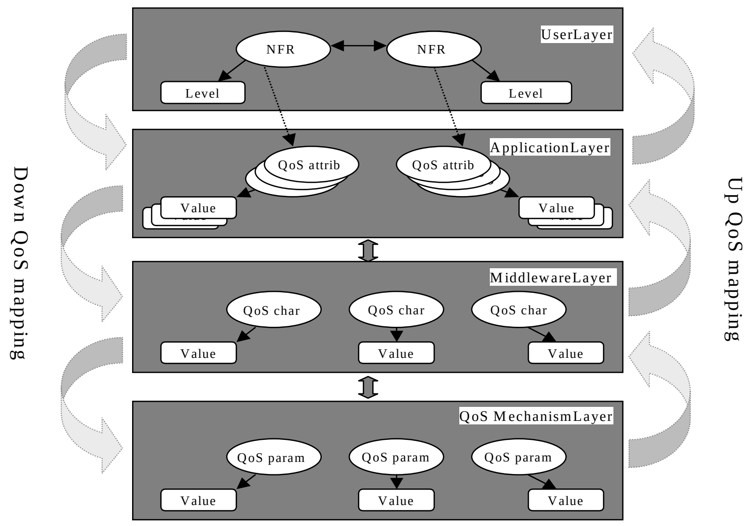


Fig. 1. Components of the ESCHER QoP Architecture

Each layer represents a particular entity together with its respective view of QoS. In the User Layer, QoS requirements consist of non-functional requirements (NFR) their associated constraint level and the relationship of priorities between them. In the Application Layer, a set of QoS attributes such as "frame rate" and "resolution" realizes NFRs defined at User Layer by considering both the category and type of application.

The third layer represents the QoS middleware view, a key architectural element for the "transparent" support of QoS. At this layer, QoS requirements are defined by QoS characteristics such as delay, jitter and loss. Finally, the lower layer is the QoS Mechanism Layer. This layer represents QoS mechanisms such as IntServ, DiffServ and ATM. Each QoS mechanism has their own QoS information represented by QoS parameters.

4.1 ESCHER Elements

There are two basic elements in ESCHER: layers and relationships between abstractions of each layer. The layers previously presented represent different QoS views (user, application, middleware and QoS mechanism). Each layer has its own abstraction to specify QoS requirements.

The relationships are classified into direct and indirect. The first one relates directly two QoS abstractions. For instance, the direct relationship namely *Realization* relates a specific NFR to the QoS attributes that realize it. The second one is composed by at least two direct relationships. For instance, the relationship between NFR and QoS characteristic is composed by *Realization* and *Affect* direct relationships. Relationships serve as basis for the QoS mapping as they model constraints imposed on mapping of user QoS requirements (NFR) into QoS parameters. They are detailed in Section 5.1.

4.2 ESCHER Operations

The ESCHER operations are grouped into two phases: configuration and monitoring. The first one is made up of the specification of user QoS requirements, the mapping of QoS requirement into QoS parameters and the negotiation for establishment of the QoS contract. The second one is composed by the activities of monitoring contracts, the reverse mapping to identify possible changes on agreed QoS level and the adaptation process.

An important characteristic of ESCHER is the facility in specifying the QoS requirements. The user basically defines the minimum set of requirements related to constraints, priorities and kind of application. The mapping process is responsible for translating the user QoS requirements into QoS parameters treated by the QoS mechanisms.

In the ESCHER, the QoS mapping is a bi-directional process, i.e., there is a *down QoS mapping*, from upper to lower layers and another one named *up QoS mapping*, in the opposite direction. Each mapping consists of the translation between QoS abstractions, respecting the relationships between them.

The down QoS mapping is a three-step process that is executed during the configuration phase. Each step is responsible for the translation between abstractions of two layers, e.g., from the user layer to the application layer.

After the down QoS mapping, it is initiated the negotiation to allocate resources to satisfy the required QoS. In the QoS Mechanism Layer, virtual resources represent an abstraction of physical resources managed by specific underlying QoS mechanisms.

The monitoring phase begins with the evaluation of the QoS provided. If the agreed QoS level is not respected, a reverse QoS mapping (up QoS Mapping) is triggered in order to identify impacts over user QoS constraints. Instead quantitative parameters, this mapping takes care of qualitative aspects when changes justify a renegotiation process.

Despite this systematic process, the user actually decides whether a renegotiation process is necessary or not. Individual quality of perception serves as basis to the user decision. The user may be satisfied with a new trade-off between performance and cost even if the QoS provided is different from the QoS initially defined.

5. Formalising the ESCHER QoP Architecture

As mentioned before, the ESCHER architecture has been formalised in Z[2]. The following points justify the adoption of the Z notation. Firstly, the emphasis of the ESCHER architecture is on the QoS information contained in contracts. That information is very complex, which demands a powerful notation for its definition. Second, the operations have to preserve the consistency of the QoS information, which can effectively be verified through the Z. Finally, the Z notation enables the formal verification of properties of the architecture, e.g., it makes possible to check whether user QoS requirements are actually realised into QoS parameters or not.

The Z specification is presented following a well-known convention: Z elements are written in italic and Z components (schemas, variables and functions) are referred to only by name.

According to the architectural components introduced in the sections 4.1 and 4.2, the ESCHER architecture (*ESCHER*) is made up of two elements, namely layers (*Layers*) and relationships (*Relationships*).

```
┌─ESCHER ─────────────────────────────────
│  Layers
│  Relationships
└─────────────────────────────────────────
```

In order to simplify the comprehension, we divide the formal specification into two parts: (1) QoS information, represented by state-schemas, and (2) QoS operations, specifically those related to the specification and the mapping process (operation-schemas). By the lack of space, only key schemas are detailed in the following.

5.1 ESCHER State Schemas

In ESCHER, each layer has its own abstraction to specify QoS requirements presented at the bottom of schema-hierarchy: non-functional requirements (*NFR*) in the user layer (*UserLayer*), QoS attributes (*QoSattr*) in the application layer (*ApplicationLayer*), QoS characteristics (*QoSchar*) in the middleware layer (*MiddlewareLayer*) and QoS parameters (*QoSparam*) in the QoS mechanisms layer (*QoSMechanismLayer*).

Each layer contains QoS specifications defined for its specific level. At user Layer, the *UserQoS* schema represents user QoS specification. In the *ApplicationLayer* the constraints imposed by the application is defined by *ApplicationQoS*.

In order to facilitate the QoS mapping process, we adopt the strategy of aggregating QoS specifications into QoS documents. There are three QoS documents considered in the ESCHER architecture: abstract QoS specification (*AbstractQoSspec*), concrete QoS specification (*ConcreteQoSspec*) and QoS contract (*ContractQoS*).

[2] Z/EVES tool was used to check the specification

In ESCHER, the middleware is responsible for QoS management service. As QoS mapping is a key task of QoS management service, the middleware layer integrates abstract and concrete QoS specifications that are manipulated during QoS mapping. The QoS mechanism layer maintains the QoS contract.

In order to understand QoS information belonging to individual layers, it is necessary to detail the basic elements *UserQoS* and *ApplicationQoS* schemas, as well as QoS documents *AbstractQoSspec*, *ConcreteQoSspec* and *ContractQoS* schemas.

Fig. 2 depicts the ESCHER state schemas. At the top of schema-hierarchy is the *ESCHER* schema, which is composed by *Layers* schema representing different QoS visions (user, application, middleware and QoS mechanism) and *Relationships* schema.

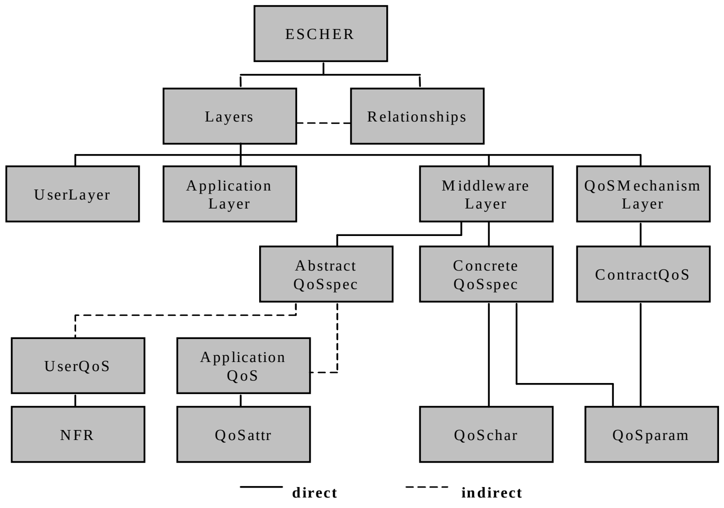


Fig. 2. ESCHER state-schemas

User QoS

The *UserQoS* schema is made up by five variables: (1) *idUser* identifies the user; (2) *QoSconstraints* defines the set of NFRs that represents QoS required by the user; (3) *QoSpriorities* establishes more restrictive NFRs; (4) *QoSperceived* represents the user quality of perception; and (5) *SatisfactionLevel* represents the satisfaction level of the user, which is related to the QoS constraints defined.

The invariant of the *UserQoS* schema includes restrictions about NFRs. If two NFRs, namely nfr_1 and nfr_2, belong to the *QoSpriorities* then only one (nfr_1 or nfr_2) can be defined as priority or more restrictive. For instance, if "quality of audio" is defined as more restrictive then "cost", this constraint must be respected in the definition of priorities. The satisfaction level of the user (*SatisfactionLevel* variable) is defined as a set having the same structure as *QoSconstraints* variable, but semantically different. The QoS constraints represent QoS desired, whilst the satisfaction level represents the user quality of perception (QoP).

```
┌─UserQoS──────────────────────────────────────────────
│  idUser: ℤ
│ QoSconstraints : (NFR ⇸Level)
│ QoSpriorities  : (NFR ×NFR) ⇸NFR
│ QoSperceived   : (NFR ⇸Level)
│ SatisfactionLevel : Level
├──────────────────────
│   ∀nfr1,nfr2:NFR | (nfr1,nfr2) ∈dom QoSpriorities ∧
│    (nfr2,nfr1) ∈dom QoSpriorities •
│   ((QoSpriorities (nfr1,nfr2) = nfr1 ∧QoSpriorities (nfr2,nfr1) = nfr1) ∨
│    (QoSpriorities (nfr1,nfr2) = nfr2 ∧QoSpriorities (nfr2,nfr1) = nfr2)) ∧
│   (dom QoSperceived ⊆dom QoSconstraints) ∧  (ran QoSpriorities ⊆dom QoSconstraints)
└──────────────────────────────────────────────────────
```

Application QoS

In the Application Layer, the *ApplicationQoS* schema represents constraints imposed by the application. It is composed by four variables: (1) *idAppl* identifies application; (2) *category* identifies application category; (3) *typeApplic* identifies the type of application; and (4) *QoSconstraints* represents constraints imposed by the application. Each QoS attribute defined in the *QoSconstraints* variable is associated with 3 values: “ideal”, “upper” and “lower".

```
┌─ApplicationQoS ──────────────────────────────────────
│  idAppl : ℤ
│ category  : Category
│ typeApplic: ApplicType
│ QoSconstraints  : QoSattr ⇸(Value ×Value ×Value)
└──────────────────────────────────────────────────────
```

Most architectural elements are directly related (see Fig. 2) to others.

Concrete QoS Specification

The concrete QoS specification (*ConcreteQoSspec*) is an internal QoS document used only by the QoS mapping process.

```
┌─ConcreteQoSspec──────────────────────────────────────
│  idUser: ℤ
│ idAppl: ℤ
│ QoSMech: QoSMechanismType
│ QoSchars: QoSchar ⇸(Value ×Value ×Value)
│ QoSparams: QoSparam ⇸(Value ×Value ×Value)
└──────────────────────────────────────────────────────
```

This QoS document is dependent on a specific QoS mechanism instantiated to support QoS guarantees. Each QoS mechanism type has its own set of QoS parameters. The *ConcreteQoSspec* schema is composed by five variables: (1) *idUser* identifies the user; (2) *idAppl* identifies the application; (3) *QoSMech* identifies the QoS mechanism used; (4) *QoSchars* is the set of QoS characteristics derived from the mapping of QoS constraints into the abstract QoS specification (user and application); and (5) *QoSparams* results from the mapping of *QoSchars* variable into a set of QoS parameters (considering a specific QoS mechanism).

QoS Contract

The QoS contract (*ContractQoS*) represents the agreement established after negotiation between the user, application and system. It is only valid when restrictions are negotiated and there is the possibility of placing resources for attending the requested services. The *ContractQoS* schema is made up five variables: (1) *idUser* identifies the user; (2) *idAppl* identifies the application; (3) *QoSrequired* represents a set of agreed QoS parameters and their respective values (ideal and boundaries); (4) *QoSprovided* represents the quality actually provided by the system; and (5) *virtualResourcesAlloc* is a set of virtual resources allocated after negotiation.

ContractQoS

idUser: $\mathbb{Z}$
idAppl: $\mathbb{Z}$
QoSrequired: *QoSparam* $\rightarrowtail$ (*Value* $\times$ *Value* $\times$ *Value*)
QoSprovided: *QoSparam* $\rightarrowtail$ (*Value* $\times$ *Value* $\times$ *Value*)
virtualResourcesAlloc: $\mathbb{P}$*VirtualResource*

Layers

As mentioned before, each layer holds QoS specification defined at specific level. In the ESCHER architecture (see Fig. 2), the middleware layer represents an architectural element for integrating the entire specification. The QoS mechanism layer is responsible for the QoS contracts.

Relationships

The relationships are key elements in the ESCHER architecture. The relationships (*Relationships*) detail the direct relationships between QoS abstract elements. The priority relationship specified inside the user QoS description is an exception. Indeed, it is the representation of individual user preferences. In order to differentiate relationships from others elements in the formal specification, we add the "*Rel*" prefix. There are eight direct relationships:

1. *RelSpecializedBy* – it relates a generic NFR and a more specific one. For example, "quality of presentation" is a generic NFR and "quality of video", "quality of audio" and "quality of synchronization" are specializations of the first one.
2. *RelConflictConvergent* – it relates two NFRs. This relationship is used to detect conflicts: the system cannot maintain the level of QoS agreed and environment changes affect two or more NFRs. If the NFRs are convergent, there is no problem. In the case of conflicting NFRs, the priorities are used to decide which NFR is more restrictive.
3. *RelRealization* – it relates a specific NFR and QoS attributes that realize it. For example, "quality of video" is a NFR realized by QoS attributes: "Frame Rate", "Aspect Ratio" and "Resolution". The QoS attributes values are associated to NFR levels.
4. *RelRelatedTo* – it relates the kind of application and a specific NFR. For example, an application like VoIP does not need to carry through NFR "quality of video". Thus, if the user specifies the generic NFR "quality of presentation" there will be established only relationships between "quality of audio" and their attributes. The

RelatedTo relationship is useful to detect inconsistency or incompleteness in the user QoS specification.

5. *RelAffect* – it relates QoS attributes and QoS characteristics. This relationship allows the detection of the impacts of changes at QoS level provided by the system. It is used by QoS adaptation mechanisms.
6. *RelClassifiedAs* – it relates category and types of applications. In the QoS paradigm, it is important to differentiate the category of the applications such as: interactive real-time and non-interactive real-time, interactive burst and asynchronous. The category classifies applications into more restrictive and more flexible ones. This relationship turns QoS specification a more friendly process, since the user does not have knowledge about the category of the application, but only about the type of the application (videoconference, distance education, etc).
7. *RelRestriction* – it relates category of application and QoS characteristics. Some restrictive categories of applications usually indicate threshold values to QoS characteristics such as delay, jitter and loss.
8. *RelQoSattrLevel* – it define the values (ideal and boundaries) associated to quality levels. For example, the QoS attribute "resolution" has different values for "high" and "low" quality levels.

5.2 ESCHER Operation Schemas

The formalization of ESCHER operations is described in the next two subsections. The first one presents the configuration phase and the second one concentrates on the monitoring phase.

5.2.1 Configuration Phase

Fig. 3 shows the activities that compose the configuration phase. As mentioned before, an important characteristic of our model is the facility of the process of specification. The user essentially defines the minimum set of requirements and the mapping process translates them into underlying QoS parameters.

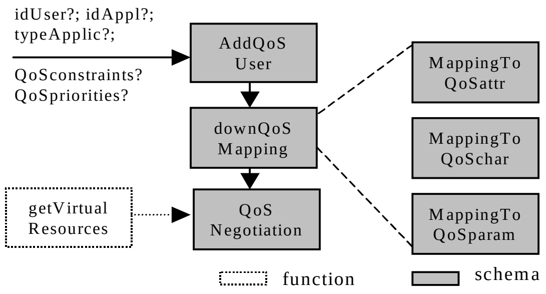


Fig. 3. The QoS specification in the configuration phase and down QoS mapping.

AddQoSUser

The *AddQoSUser* schema specifies the process of the user QoS requirement definition. It is composed by five input variables: (1) *idUser*? identifies the user; (2) *idAppl*? identifies the application; (3) *typeApplic*? identifies the kind of application; (4) *QoSconstraints* is a set of constraints imposed by the user; and (5) *QoSpriorities* identifies the NFRs considered more restrictive by the user. The category of the application is modelled through the *RelClassifiedAs* relationship.

AddQoSUser

$\Delta Layers$
$\Xi Relationships$
$idUser?: \mathbb{Z}$
$idAppl?: \mathbb{Z}$
$typeApplic?: ApplicType$
$QoSconstraints?: NFR \rightarrow Level$
$QoSpriorities?: NFR \times NFR \rightarrow NFR$

$\exists u: UserQoS;\ a: ApplicationQoS$
$\quad \mid\ u\,.\,idUser = idUser? \ \wedge u\,.\,QoSconstraints = QoSconstraints?$
$\quad \wedge\ u\,.\,QoSpriorities = QoSpriorities?$
$\quad \wedge\ a\,.\,idAppl = idAppl? \ \wedge a\,.\,typeApplic = typeApplic?$
$\quad \wedge\ a\,.\,category = RelClassifiedAs\ typeApplic?$
$\quad \wedge\ a\,.\,QoSconstraints = \emptyset$
$\quad \bullet\ u \notin UserLayer \ \wedge a \notin ApplicationLayer \ \wedge UserLayer' = UserLayer \cup \{u\}$
$\quad \wedge\ ApplicationLayer' = ApplicationLayer \cup \{a\} \ \wedge MiddlewareLayer' = MiddlewareLayer$
$\quad \wedge\ QoSMechanismLayer' = QoSMechanismLayer$

downQoSMapping

The down QoS mapping (*downQoSMapping*) is a three-step process composed by: (1) the translation of NFR into a set of QoS attributes (*MappingToQoSattr*); (2) the translation of the set of QoS attributes into a set of QoS characteristics (*MappingToQoSchar*); and (3) the translation of the set of QoS characteristics into a set of QoS parameters, by considering a specific QoS mechanism (*MappingToQoSparam*). Each mapping respects the relationships between QoS abstractions. A generic function (*getVirtualResources*) was defined to represent allocation of virtual resources.

QoSnegotiation

The process of QoS negotiation (*QoSnegotiation*) is based on both the resulting parameters of the down QoS mapping (maintained in the *concreteQoSspec*) and a particular policy of admission control (the variable *AdmissionPolicy*), which is defined in an abstract manner. This process automatically triggered during the configuration phase. The allocation treats with virtual resources obtained through a specific function. As we mentioned previously, the management of physical resources is out of scope of our proposal. A generic function (*getVirtualResources*) has been defined to represent allocation of virtual resources.

5.2.2 Monitoring Phase

The monitoring phase (Fig. 4) has a dynamic nature as it occurs at run-time. This phase is composed by five specific activities: the *MonitoringContract* is executed by protocols of lower level such as RSVP (ReSerVation Protocol); (2) the *upQoSMapping* is a reverse mapping that reflects the changes along the layers; (3) the *UserSatisfactionLevel* is used to get the level of satisfaction of the user; (4) the *noAdaptationProcess* stops the renegotiation process; and (5) the *AdaptationProcess* starts a renegotiation process.

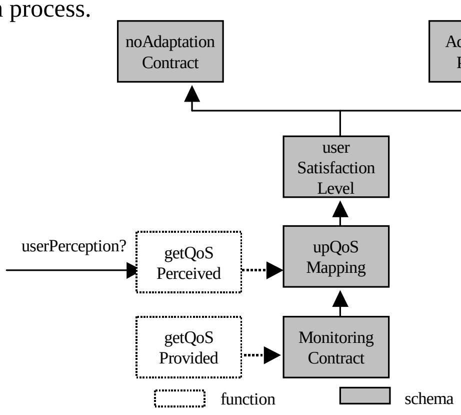


Fig. 4. The Monitoring phase and up QoS mapping

Additionally, the monitoring phase uses some generic functions to abstract tasks executed by underlying mechanisms. These generic functions are divided into two kinds: external and internal. Either the users or the underlying mechanisms use the first one. The second one is used by the ESCHER operations.

The purpose of monitoring contract activities is to get the QoS provided for the system and to verify if contracts are being respected. If agreed values are not respected, a reverse mapping (*upQoSMapping*) is triggered. It reflects the changes along layers, from QoS Mechanism Layer to User Layer, and aims to evaluate the impacts of changes in the level of QoS agreed over QoS constraints defined by the user. Additionally, a generic external function (*getQoSprovided*) gets the level of QoS provided. The difference between QoS required and QoS provided defined at QoS contract is used to identify the user satisfaction level.

The adaptation process (*AdaptationProcess*) is specified by the conjunction of *AdaptationContract*, *downQoSMapping* and *QoSnegotiation* schemas. The process of contract adaptation (*AdaptationContract)* tries to define a future adjustment of constraints, mapping and renegotiation for new resource allocation. It takes into account the new level of QoS provided and the user satisfaction level. If more than one NFR is affected and conflicting (e.g., Audio Quality and Cost), priorities defined by the user are used to solve the conflict.

6. An Example: Videoconference

In order to illustrate our approach, we have modelled a videoconference application. The main objective of this example is to show how the QoS abstractions are related and how the mapping can be realized. In Fig. 5, it is possible to observe the QoS mapping process along ESCHER layers.

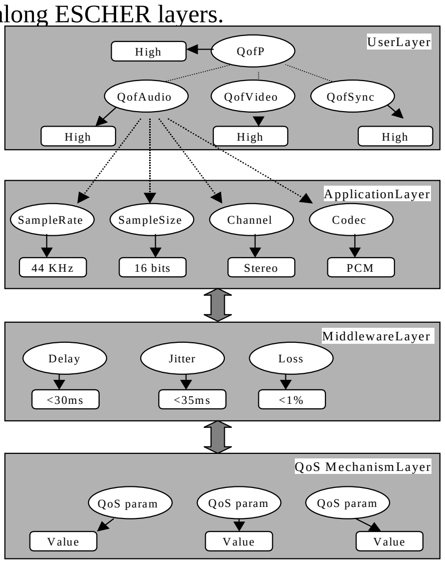


Fig. 5. Modelling of Videoconference Application

In this example, the user defines a general NFR called "quality of presentation" (QofP). The relationship *RelSpecializedBy* is used to translate this NFR into more specific ones to represent quality of audio (QofAudio), quality of video (QofVideo) and quality of synchronization (QofSync). The mapping of quality of audio is presented. It has been considered usually recommended set of QoS attributes at Application Layer and values associated to QoS characteristics at Middleware Layer. QoS parameters at QoS Mechanism Layer are treated in an abstract way.

7. Conclusions and Future Work

We have presented a formal QoP architecture, namely ESCHER, which explicitly takes into account the user perception, whilst it also proposes a systematic mapping of QoS requirements (at user level) into QoS parameters (treated by QoS mechanisms).

In addition to the high level of abstraction the user QoS requirements are specified, the benefits of our proposal also include: the separation of concerns in QoS specification and the treatment of user satisfaction as a trade-off among NFRs. The first one allows legacy applications to use QoS services offered by the middleware. The second one serves as a basis for more flexible QoS adaptation mechanism.

In terms of future work, we intend to concentrate on the QoS mapping by identifying and formalizing the rules to make this process automatic. We also intend to verify some properties of our model. For example, the capacity of our model to reflect changes in the level of provided QoS and vice-versa.

References

1.P.Bernstein."Middleware, a Model for Distributed System Service", Communications of the ACM, 39:2, February 1996

2.P.E. Pedersen, L.B. Methlie and H. Thorbjørnsen, "Understanding mobile commerce end-user adoption: a triangulation perspective and suggestions for an exploratory service evaluation framework". HICSS-35, Hawaii, US, Jan 7-10, 2002.

3.C. Aurrecoechea, A. Campbell and L. Hauw "A Survey of Quality of Service Architecture", Multimedia Systems Journal, November, 1995.

4.G. Ghinea and J.P. Thomas. "QoS Impact on User Perception and Understanding of Multimedia Video Clips", Proc. of ACM Multimedia '98, Bristol, United Kingdom, 1998

5.EURESCOM, "Offering Quality Classes to end users", Deliverable 1,Volume 1 of 2, Project P906-GI, QUASIMODO, http://www.eurescom.de/public/projectresults/results.asp

6. J.M. Spivey. The Z notation: A reference Manual, Second Edition. Prentice Hall, 1992

7.T.Yamazaki and J. Matsuda, "On QoS Mapping in Adaptive QoS Management for Distributed Multimedia Applications", Proc.ITC-CSCC' 99, vol.2, pp. 1342-1345, July, 1999.

8.Klara Nahrstedt, Dongyan Xu, Duangdao Wichadakul and Baochun Li. "QoS-Aware Middleware for Ubiquitous and Heterogeneous Environments" IEEE Communications Magazine, 39(11), 2001.

9.Kachroo V, Karr DA, Rodrigues C, Loyall JP, Schantz RE, Schmidt DC, "Integration of QoS-Enabled Distributed Computing Middleware for Developing Next-Generation Distributed Applications". Proc. of the ACM SIGPLAN (OM 2001), June 18, 2001, Snowbird Utah.

10.N. Bhatti, Anna Bouch, and Allan Kuchinsky – "Integrating user-perceived quality into web server design". In 9th International World Wide Web Conference, Amsterdam, May 2000.

11.Anna Bouch, Allan Kuchinsky, Nina Bhatti - "Quality is in the Eye of the Beholder: Meeting Users' Requirements for Internet Quality of Service" – Proc. of the CHI 2000 Conference on Human Factors in Computing Systems, p.297-304, April 01-06, 2000, Netherlands

12.ETSI, TS 101329-2 v1.1.1, Telecommunication and Internet Protocols Harmonisation over Networks (Tiphon); End to End Quality of Service in TIPHON Systems; Part 2: Definition of Quality of Service (QoS) Classes, 2000-07, www.etsi.org

13.I. Widya, R.E. Stap, L.J. Teunissen, B.F. Hopman "On the end-user QoS-awareness of a distributed service environment". Accepted at 6th PROMS' 01. Netherlands, October 2001

ServiceFORGE: A Software Architecture for Power and Quality Aware Services

Radu Cornea, Nikil Dutt, Rajesh Gupta*, Shivajit Mohapatra, Alex Nicolau, Cristiano Pereira*, Sandeep Shukla**, Nalini Venkatasubramanian

University of California, Irvine, * University of California, San Diego
** Virginia Tech. University, Blacksburg, Virginia

Abstract. We present a novel power management service in QoS-brokerage architecture that relies on multi-level middleware services to act as brokers in delivering multimedia content in distributed real-time systems under performance and power constraints. To minimize power consumption, the power management service provides for minimal interaction among the system components as well as provides dynamic control of speed scaling (such as voltage, frequency) and component shutdown. These power control “knobs” enable the service to match instantaneous application needs against available energy resources. The power management service is implemented using the Comp|OSE middleware framework using a power-aware API that allows the applications programmer to ensure a continuous dialogue between the distributed embedded systems and changing application needs. In this paper, we focus on the interfaces to the power management service that ensure semantically correct composability of power management actions in a distributed systems and its use in applications programming. We present example of a multimedia streaming server to handheld devices that demonstrates the use of automatic speed scaling knobs in minimizing energy consumption.

1. Introduction

Power management is important for a range of embedded applications from portable terminals to ad hoc sensor networks. The focus of much of the work in this area, so far has been on minimization of energy at the node level [FengSechrest96, ChandraVahdat02]. For distributed embedded systems, the problem is much more complex because of the dynamic tradeoffs involved at several levels of abstraction between local processing and communication/coordination [Yuan et al 2003]. The correct way to think about it is to treat as a multi-level ‘service’ in a distributed embedded system [Cornea et al 03]. In this paper, we describe a software system architecture that enables the system architect to compose distributed power/performance related decision making and to ensure compliance with functionality and system energy constraints based on the runtime conditions. This is done by means of “brokers.” These brokers are built in a model-based system specification that allows reasoning about the functional and non-functional properties of the system from the properties of the constituent components and the composition mechanism applied[VenkatasubramanianTalcottAgha01]. We use a middleware infrastructure that lends itself to platform specific optimization for performance and size. Specifically, we focus on adaptive and reflective middleware services [Venkatasubramanian et al 2001] to meet the application requirements and to dynamically smooth the imbalances between demands and changing environments. **Fig. 1** illustrates the fundamental levels of adaptation and reflection supported by middleware services: (a) changes in the middleware, operating systems, and networks beneath the applications to continue to meet the required service levels despite changes in resource availability, such as changes in network bandwidth or power levels, and (b) changes at the application level to either react to currently available levels of service or request new ones under changed circumstances, such as changing the transfer rate or resolution of information over a congested network. In both instances, the middleware must determine if it needs to (or can) reallocate resources or change strategies to achieve the desired QoS. Embedded applications must be built in such a way that they can change their QoS demands as the conditions under which they operate change. Mechanisms for reconfiguration need to be put into place to implement new levels of QoS as required, mindful of both the individual and the aggregate points of view, and the conflicts that they may represent.

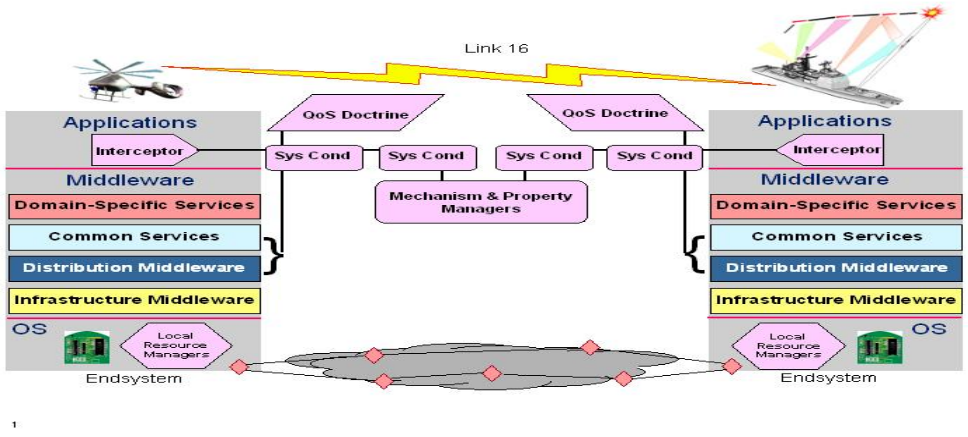


Fig. 1. Middleware Services and QoS Brokerage.

2. ServiceFORGE: A Software Architecture for Quality-Aware Services

We approach the application development in ServiceFORGE as not only specification of the desired functionality but also specification and management of a contract with the underlying mobile software system on timing performance and energy needs. The underlying software is aware of the finite energy/power resources and makes use of its own contract and brokering services to adapt its functional needs to match low-level computing, communication and networking capabilities. Our strategy is based on building an application development framework that allows radically enhanced configurability and adaptability in pretty much all aspects of software and networking processes. These include reconfiguration algorithms that exploit adaptability in the various tasks that constitute the wireless protocols and management of application-level tasks to ensure efficient use of the dynamically changing battery resource as well as performance and energy requirements of the executing tasks. Our approach consists of two parts: power aware operating and runtime system services, and their interface to middleware services that ensure efficient brokerage of the application needs and system resource constraints.

Power Aware Operating and Runtime System Services

For effective system-level power management, it is important that an application is able to monitor and control both electrical knobs (such as voltage and scheduling), as well as exercise control on the task scheduling and shutdown states of different parts of the client device (node). The operating system is an important place for making power management decisions since it has the knowledge about timing, usage and traffic patterns of applications. We modify traditional OS services to make them power-aware vis-à-vis their execution so that energy is used in an efficient fashion. At the system-level, power management consists of three parts: (a) system components as *resources* that provide service such a computation or I/O and consume power, (b) application functionality as *tasks* that utilize the resources, and (c) system timing performance and result quality requirements as *service contracts*. Each resource may have one or more *operating modes* (e.g. a CPU core may be in run or shutdown mode, and a radio subsystem may be in transmit or receive or idle mode). The transitions between operating modes of a resource are dictated by the availability of tasks needing that resource, and the functional requirements of those tasks (e.g. a packet sender task would put the radio in the transmit mode). Further, a resource in each of these modes can be put in one of multiple operating points in a mode-specific *power-speed subspace* (and more generally *power-speed-quality subspace*) via “control knobs” that could be relatively universal or perhaps resource-specific. An example of a relatively universal control knob is the processor supply voltage. Supply voltage change, with a coordinated change in the clock frequency, leads to multiple power-vs.-speed points. The transitions between the power-speed-quality operating points will be dictated by the system timing performance and result quality requirements [Choi02]. At any given instant the system may be viewed to be in a specific *power state* corresponding to a specific permutation of the operation mode and power-speed-quality operating point for each resource. The key to system-level power/performance optimization is a power management control strategy consisting of task and resource specific algorithms to decide the power state evolution of the system. We describe our approach to control node-specific power/performance constraints in the next section, followed by a description of their integration with the middleware brokerage services using the example of a streaming video server.

3. Power Aware Nodal Services in ServiceFORGE

As mentioned earlier, the chief goal of power awareness at the node level is to enable **a continuous dialogue between the application, the OS, and the underlying hardware**. This dialogue establishes the functionality and performance expectations (or even contracts, as in real-time sense) within the available energy constraints. We describe here our implementation of a Power Aware Software Architecture (PASA). PASA is composed of two software layers and the RTOS kernel. One layer interfaces applications with operating system and the other layer makes power related hardware "knobs" available to the operating system. Both layers are connected by means of corresponding power aware operating system services as shown in **Fig. 2**. At the topmost level, embedded applications call the API level interface functions to make use of a range of services that ultimately makes the application energy efficient in the context of its specific functionality. The API level is separated into two sub-layers. PA-API layer provides all the functions available to the applications, while the other layer provides access to operating system services and power aware modified operating system services (PA OS Services). Active entities that are not implemented within the RTOS kernel should also be implemented at this level (threads created with the sole purpose of assisting the power management of an operating system service, such as a thread responsible for killing threads whose deadlines were missed). We call this layer the power aware operating system layer (PA-OSL).

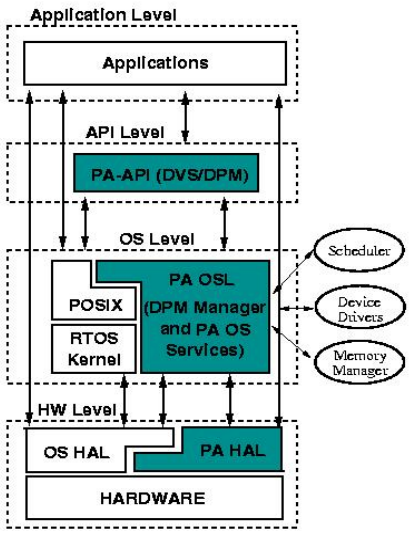


Fig. 2. Nodal Services and API for Power

To interface the modified operating system level and the underlying hardware level, we define a power aware hardware abstraction layer (PA-HAL). The PA-HAL gives the access to the power related hardware ``knobs" in a way that makes it independent of the hardware.

Table below lists the functions relevant to the implementation of power aware scheduling techniques. At the PA-API layer there are functions to create types (informing the real time related parameters) and instances of tasks, to notify start and end of tasks (needed by the OS in order to detect whether the task execution is over and the deadline of a task has been met), and to either inform the application about the execution time predicted by the OS or tell the OS about the execution time prediction estimated by the application (which can be based on application specific parameters). At the PA-OSL layer there are functions to manipulate information related to the power aware scheduling schemes that are maintained within the kernel (such as the type table in the case of the scheduler), the thread responsible for killing threads whose deadlines were missed (assuming that the threads whose deadlines were missed are no longer useful). The alarm handler notifies the killer thread, which in turn kills the thread and re-creates it. The overhead of having an extra thread is minimal since the killer thread is constantly blocked unless a request to kill another thread is received. When it happens, the killer thread wakes up and finishes the execution of the proper thread. At the PA-HAL layer functions to manipulate processor frequency and voltage levels and low power states are present. These are called by the RTOS scheduler when slowing down the processor or shutting it down. For processor frequency and voltage scaling, different platforms have different precautions that have to be taken care of before doing the scaling. These precautions might have to be done before the scaling, after it or both before and after. For these the functions pahal_pre_set_frequency_and_voltage and pahal_post_set_frequency_and_voltage are provided and must be implemented by the OS programmer according to the platform. And finally functions to poll the status of battery based platforms are also important in order to enhance their lifetime.

Layer	**Function name**
PA-API	paapi_dvs_create_thread_type(), paapi_dvs_create_thread_instance() paapi_dvs_app_started(), paapi_dvs_get_time_prediction() paapi_dvs_set_time_prediction(),paapi_dvs_app_done(), paapi_dvs_set_adaptive_param(),paapi_dvs_set_policy(), paapi_dpm_register_device()
PA-OSL	paosl_dvs_create_task_type_entry(),paosl_dvs_create_task_instance_entry(), paosl_dvs_killer_thread(),paosl_dvs_killer_thread_alarm_handler(), paosl_dpm_register_device(), paosl_dpm_deamon()
PA-HAL	pahal_dvs_initialize_processor_pm(), pahal_dvs_get_frequency_levels_info() pahal_dvs_get_current_frequency(), pahal_dvs_set_frequency_and_voltage() pahal_dvs_pre_set_frequency_and_voltage(), pahal_dvs_post_set_frequency_and_voltage() pahal_dvs_get_lowpower_states_info(), pahal_dvs_set_lowpower_state() pahal_dpm_device_check_activity(), pahal_dpm_device_pre_switch_state() pahal_dpm_device_switch_state(), pahal_dpm_device_post_switch_state() pahal_dpm_device_get_info(), pahal_dpm_device_get_curr_state() pahal_battery_get_info()

The piece of code that follows shows an example on how the PA-API functions are used in a MPEG decoder source code when creating threads using PA-API functions. A thread is created specifying that the deadline and period are 100 and the worst case execution time is 30 (assuming it was profiled and therefore known ahead of time. The thread is instantiated and access to the power-aware functionality contracts is enabled and terminated by the functions paapi_app_started() and paapi_app_done() respectively. These functions delimit the work done by the threads which is encapsulated in one single function in this example.

```
void main() {
  mpeg_decoding_t =
    paapi_create_thread_type(100,30,100);
  paapi_create_thread_instance(mpeg_decoding_t,
mpeg_decode_thread); }
```

```
void mpeg_decode_thread() {
  for (;;) {
    paapi_app_started();
    /* original code */
    mpeg_frame_decode()
    paapi_app_done();   }}
```

This provides a generic dynamic power management (DPM) API sufficient to support different devices and DPM policies by using a common set of functions. The API also provides a common framework for implementing new (DPM) policies. For DPM purposes, each device is registered with the power manager and with each device we attach enough information to execute whichever policy the device was registered to be managed with. Often device DPM techniques switch devices to low-power modes or states based on how long the device has been idle. For instance, threshold values are defined for each device so that the longer the device is idle, the deeper the sleep mode it is switched to. A common set of functions and data structures have to be defined in order to manage such devices. These furnish the implementation of the DPM techniques and provide the guidelines for implementing new ones. Some of the functions defied for this purpose are listed in the previous table and are shortly described below:

- `dpm_device_check_activity()` - This function finds out whether the device was activated or has been idle since the last time it was queried. To do that, a device activity structure has to be kept and has to be compared against a new activity information every time the device is queried (on the embedded Linux platform, for instance, this information comes from the /proc virtual file system interface. For other operating systems without such interface this information has to be kept by the API in form of tables in order to track the device activity). If the amount of activity is the same it means that nothing has happened since the last time it was checked. Otherwise some activity has happened and the stored information is updated. This function makes sense only if the policy used is based on the activity information (kept in a device status table).
- `dpm_device_switch_state()` - This function will switch the state of the device from origin state to destination state.
- `dpm_device_get_info()` - It gets information about the device.
- `dpm_device_get_state()` - It tells in which state the device is in currently.
- `dpm_device_register()` - It registers the device along with the appropriate functions and power management policy to handle it. This information is kept in the device info table.

Structures containing the possible states each device can be at, as well as which policies, are attached to each device and the information needed to implement such policies are defined and kept within the kernel. The kernel DPM entity consists of tasks associated with each device and implementing a specific policy for managing the device. If all devices use the same policy then multiple instances of this policy are created and they manage each device individually.

The piece of code below shows a threshold based dynamic power management scheme. For each device state there is an associated threshold which defines when to switch to that state as described in the lower envelope algorithm.

```
void threshold_policy_deamon(device_info_t dev){
  unsigned idleness;
  for (;;) {
    /* check for how long the device has been idle */
    idleness = dev->check_activity(dev);
    /* if idle for longer than the threshold
       switch to   next state */
    if ( idleness > dev->check_state()->threshold ) {
      dev->check_state()->switch_state(dev,
         dev->check_state, dev->check_state()->next); }
    /* sleep until next period for checking idleness */
    sleep(dev->policy_info->th_policy->period); }}
```

When mixing DPM and DVS algorithms in the same platform there is a tradeoff on whether to slowdown as much as possible or to execute some tasks faster than the minimum possible frequency and rearrange the idle times in order to get better changes to shutdown some of the system components. In [IraniShuklaGupta03], we have devised an algorithm to optimize this tradeoff on a system. The algorithm, named as *Procrastinator*, adjusts (or procrastinates) start times and deadlines of some of the tasks in order to create longer idle times and obtain more chances to wisely bring the device to a low power mode. The PASA architecture and its API make it possible to utilize the combined DVS and DPM opportunities and improve their effectiveness with additional information available from the application itself.

4. Middleware Power/Performance Brokerage Service in ServiceFORGE

Node level dynamic power management described in the previous section is only a first step in achieving an application-level control of the system power/performance tradeoffs. We approach the distributed power/performance optimization problem as one of middleware services that interact with the node level services to make the right tradeoffs in the context of application behavior and its needs [Mohapatra et al 03, MohapatraVenkatasubramanian03].

To explain this, let us use the example of a multimedia streaming from a server to a set of mobile handheld devices. The system architecture shown in **Fig. 3** below consists of a multimedia server, a proxy server (that adapts the video stream to client capabilities), a wireless access point and the clients (low-power wireless devices). The multimedia server streams videos to clients on request from users. All communication between the servers and clients are routed through the proxy server, which can transcode the video stream in real time.

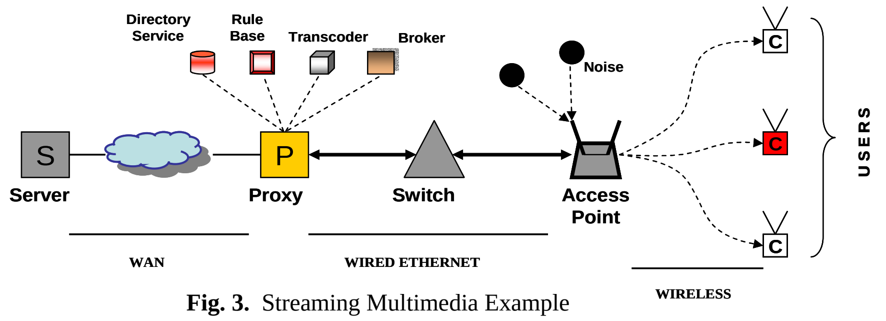


Fig. 3. Streaming Multimedia Example

Middleware level components (services) execute on both the handheld device and the proxy, performing two important functions: On the device, it sends residual energy availability information the proxy and relates video stream parameters and network control information to lower abstraction layers. This information is conveyed using the PASA API for the HAL/OSL layers. On the proxy, it performs a feedback based power aware admission control and real time transcoding of the video stream based on the feedback from the device. It also controls the video transmission over the network based on the load on the network and video stream quality level. To illustrate the application control of the client power and fairness of the service to multiple clients, we use quality level of the video (specified as PSNR).

Fig. 4 shows the overall ServiceFORGE architecture. To implement the desired level of power/performance control, we assume that the following levels of abstraction apply to both the proxy and the clients: architecture, operating system, middleware, application. Each of these levels has components/services interacting with corresponding services on the same level or with components at a different level of abstraction. The architecture level includes most of hardware components: CPU with memory, display, network card, etc. OS level provides the scheduler, DVS control, power-aware APIs and other OS level services. The middleware level provides a series of services, like network management, transcoding, admission control, mobility information, etc. The video application runs at the application level, with the other tasks running on the device.

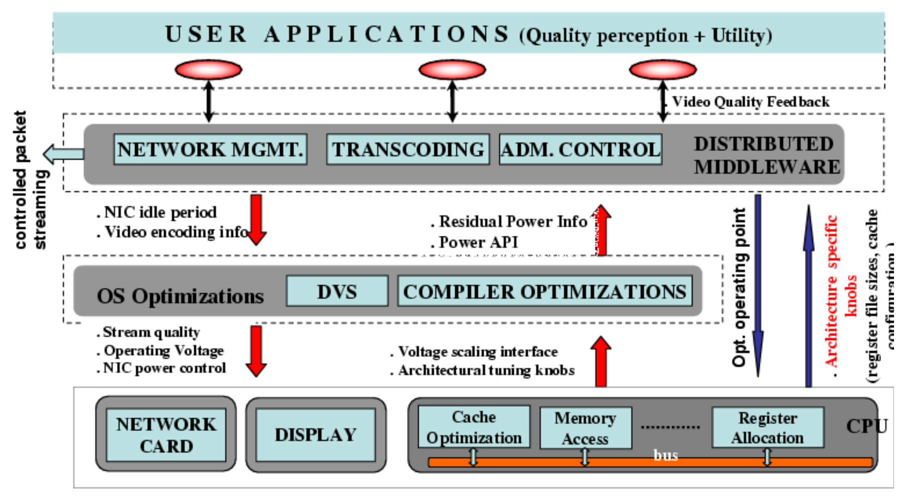


Fig. 4. Architecture of ServiceFORGE

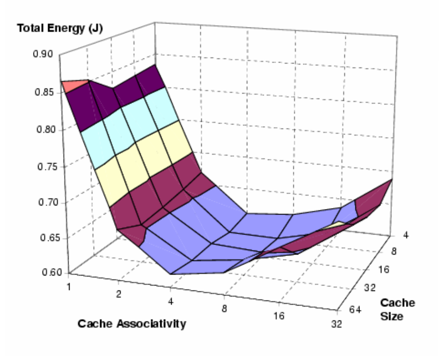
Components and services at different level of abstraction interact together with a final goal of improving the overall system performance, including power, deadlines and quality of service. There are various control knobs available at the device: CPU voltage and frequency scaling, memory system configuration, network card access pattern. The video stream can be controlled through its encoding parameters: frame rate, bitrate, and frame rate. Each of these parameters are controlled by the middleware services and adapted as required by the runtime conditions in the system. For example, if the residual energy available at a mobile device drops, the control decreases the quality of the video stream by lowering its frame rate or one of the other parameters (frame size, bitrate). Similarly, when a new user joins the system, resources need to be freed in order to accommodate the new node in the network (allocate network bandwidth, transcoder CPU time, etc.).

The middleware services also control network transmission. To save power, video stream data is grouped into short burst transmissions and sent periodically over the network. This allows the network card at the device to go into longer periods of low-power sleep mode [Shenoy03].

Each component in the abstraction hierarchy provides services to the other components on the same node or on other nodes in the network. During runtime, there is a continuous exchange of information and control between nodes to ensure that the constraints imposed on the system are met and quality of service is preserved for all the clients. If for some reasons these conditions cannot be satisfied the admission control component may decide to renegotiate video quality levels with all users in the system.

Experiments Using Video Server Example

We performed several experiments to evaluate power savings and performance improvements at different levels of abstraction as well as globally for the entire system. At the architecture level, we selected cache parameters as optimizing knobs and profiled video clips for a large space of cache configuration points. **Fig. 5** shows results from configuring the data cache to meet the requirements of particular video streams. These changes alone can yield 10-15% in power savings. Combining frequency and voltage scaling with cache reconfiguration increase the opportunities for power savings, as the processor can be run at a lower voltage and frequency when decoding less complex frames. This combined approached yields up to 60% in energy savings as compared with the initial architecture.



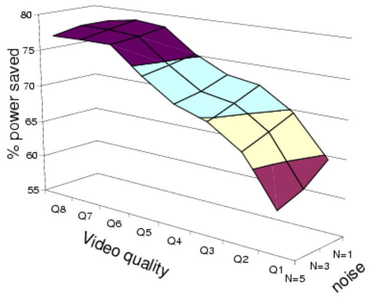


Fig. 5. Cache Optimization Search Space

Fig. 6. Power Savings in the Network Card

Fig. 6 shows that at the network level we obtained up to 70% power savings by sending optimized bursts of video and turning the network interface off (sleep mode) between consecutive bursts. The ideal burst time was computed for each quality level and for different network load - users in the network are modeled as noise. Finally, we evaluate the performance of the integrated framework. Our goal is to provide an optimal user experience and maintain an acceptable utility factor for the system. We define an "*acceptable utility factor*" to be obtained when the system can stream the highest possible quality of video to the user such that time, acceptable quality and power constraints are satisfied (i.e the video clip runs to completion, at a quality level above or equal to the one the user specified, the difference between the two defining the final utility factor). To accomplish this it is important to understand the notion of video quality for a handheld device and its implications on power consumption. **Fig. 7** shows how adaptive middleware provided by ServiceFORGE can improve the utility factor for the integrated framework.

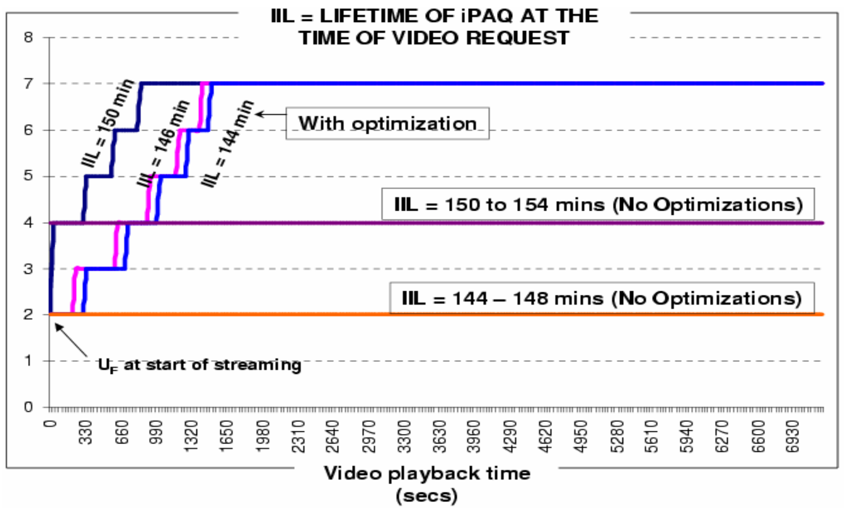


Fig. 7. Utility Factor over Time.

5. Summary and Conclusions

Ensuring best system performance in presence of very real resource constraints in distributed embedded systems is a difficult problem. Solving this problem requires analysis of available system resources, application needs in presence of dynamically changing operating conditions. Instead of seeking optimization techniques, our approach is to enable application participation with the runtime systems in setting the appropriate resource utilization policies. Towards that end, we have built Service-FORGE to provide two basic capabilities: capability for the middleware to carry out a dialogue with the application in determining its needs and conveying these through a structured interface to individual nodes; and the capability for the individual nodes to change performance/power usage knobs based on the middleware directives. Early experiments suggest that this architecture can be useful in achieving better quality of results for the same power budgets in the case of streaming video. Additional experimentation across various application domains is necessary to understand how application programming can be structured to take advantage of the new services in the system software.

References

[ACEORB] Center for Distributed Object Computing, "The ACE ORB (TAO)" www.cs.wustl.edu/_schmidt/TAO.html Washington University.

[aspectGAMMA02] M. Mousavi, G. Russello, M. Chaudron, M.A. Reniers, T. Basten, A. Corsaro, S. Shukla, R. Gupta, D. Schmidt, "Aspects+GAMMA=AspectGAMMA: A Formal Framework for Aspect-Oriented Specification", presented at the Early Aspects Workshop, Twente, Netherlands, April 2002.

[Balboa] Balboa Project. Component Composition Enviornement Home page: http://www.cecs.uci.edu/_balboa.

[Banatre93] Jean-Pierre Banatre and Daniel Le Metayer, Programming by multiset transformation, Communications of the ACM (CACM), 36(1):98--111, January 1993.

[Bapty et al, 2000] Bapty T., Neema S., Scott J., Sztipanovits J., Asaad S, "Model-Integrated Tools for the Design of Dynamically Reconfigurable Systems", VLSI Design, 10, 3, pp. 281-306, 2000.

[Birkhoff1933] G. Birkhoff. On the Combination of Subalgebras. Proceedings of Cambridge Philosophical Society, 1933.

[Blair et al 98] Gordon S. Blair, G. Coulson, P. Robin, and M. Papathomas, "An architecture for next generation middleware," in Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing, Springer-Verlag, London, 1998.

[Boehm80] Boehm, B. Software Engineering Economics, Prentice Hall, 1980.

[Bol00] Bollella, G., Gosling, J. "The Real-Time Specification for Java," Computer, June 2000.

[Booch98] Grady Booch, Ivar Jacobson, James Rumbaugh, Jim Rumbaugh "The Unified Modeling Language User Guide", The Addison-Wesley Object Technology Series, 1998.

[BroyKrueger98]	M. Broy, I. Krüger: Interaction Interfaces - Towards a scientific foundation of a methodological usage of Message Sequence Charts, in: J. Staples, M. G. Hinchey, Shaoying Liu (eds.): Formal Engineering Methods (ICFEM'98), IEEE Computer Society, 1998
[BroyStoelen01]	M. Broy, K. Stølen: Specification and Development of Interactive Systems. Focus on Streams, Interfaces, and Refinement. Springer, 2001
[ChandraVahdat02]	S. Chandra and A. Vahdat. "Application-specific Network Management for Energy-aware Streaming of Popular Multimedia Formats". In Usenix Annual Technical Conference, June 2002.
[Chaudron98]	Chaudron, M. R. V, "Separating Computation and Co-ordination in the Design of Parallel and Distributed Systems", Ph.D thesis, Leiden University, 1998.
[Chaudron94]	Michel R.V. Chaudron, Schedules for Multiset Transformer Programs, Technical Report no 94-36, Department of Computer Science, Leiden University, December 1994.
[Choi02]	K. Choi, K. Dantu, W.-C. Chen, and M. Pedram. "Frame-Based Dynamic Voltage and Frequency Scaling for a MPEG Decoder". In ICCAD 2002.
[Chou94]	P. Chou, and G. Boriello, " Software Scheduling in the co-synthesis of Reactive Real-Time Systems", in Proceedings of the 31st Design Automation Conference, 1994.
[Corba 2000]	Object Management Group, "The Common Object Request Broker: Architecture and Specification, 2.4 ed.", Oct. 2000.
[Cornea et al 03]	R. Cornea, N. Dutt, R. Gupta, I. Krueger, A. Nicolau, D. Schmidt, S. Shukla, "FORGE: A Framework for Optimization of Distributed Embedded Systems Software", IPDPS 03.
[Culler et al 01]	David E. Culler, Jason Hill, Philip Buonadonna, Robert Szewczyk, and Alec Woo, "A Network-Centric Approach to Embedded Software for Tiny Devices",in DARPA workshop on Embedded Software, 2001.
[Donahue et al 2001]	S. M. Donahue, M.P. Hampton, M. Deters, J. M. Nye, R.K. Cytron, and K. M. Kavi, "Storage allocation for real-time, embedded systems," in Embedded Software: Proceedings of the First International Workshop (T.A. Henzinger and C.M. Kirsch, eds.), pp.131-147.
[Donahue et al 2002]	S. Donahue, M. Hampton, R. Cytron, M. Franklin, and K. kavi, "Hardware support for fast and bounded-time storage allocation," Second Annual Workshop on Memory Performance Issues (WMPI 2002), 2002
[Doucet-date02]	F. Doucet, R. Gupta, M. Otsuka, S. Shukla, "An Environment for Dynamic Component Composition for Efficient Co-Design", Accepted for presentation at the Design Automation and Test Conference (DATE 2002), Match 2002.
[Eme90]	E. Allen Emerson. Temporal and Modal Logic. In Jan van Leeuwen, editor, Handbook of Theoretical Computer Science, volume B, pages 995--1072. Elsevier, 1990.
[Esterel Tech]	Esterel Technologies Web Page, http://www.esterel-technologies.com/
[FengSechrest96]	W.chi Feng and S. Sechrest. "Improving data caching for software mpeg video decompression". In IS&T/SPIE Digital Video Compression: Algorithms and Technologies, 1996.
[FinkbeinerKrueger01]	B. Finkbeiner, I. Krüger: Using Message Sequence Charts for Component-Based Formal Verification. Specification and Verification of Component-Based Systems (SAVCBS). Workshop at OOPSLA 2001.

[Frappier 2000] Marc Frappier, Henri Habrias , “Software Specification Methods : An Overview Using a Case Study (Formal Approaches to Computing and Information Technology”, Springer Verlag, November 2000.

[Gal01] Andreas Gal, Wolfgang Schroder-Preikschat, and Olaf Spinczyk, “On Aspect Orientation in Distributed Real-Time Dependable Systems”, "On Aspect-Orientation in Distributed Real-time Dependable Systems", Accepted at the Seventh IEEE International Workshop on Object-oriented Real-time Dependable Systems (WORDS 2002) , San Diego,CA, January 7-9, 2002

[GarlanAllenOckerbloom95] D. Garlan, R. Allen, and J. Ockerbloom. Architectural Mismatch: Why Reuse Is So Hard. IEEE Software, November 1995.

[Genssler et al 2002 T. Genssler, O. Nierstraszand B. Schoenhage. Componenets for embedded software: The pecos approach. In Proc. Int. Conf. On Compilers, Architecture, and Systhesis for Embedded Systems, 2002.

[Gill et al 2001] Chris Gill, David Levine, and Douglas C. Schmidt, “The Design and Performance of a Real-Time CORBA Scheduling Service,” The International Journal of Time-Critical Computing Systems, special issue on Real-Time Middleware, guest editor Wei Zhao, Volume 20, Number 2, March 2001.

[GorlickRazouk91] M. M. Gorlick and A.R.R. Razouk. Using weaves for software construction and analysis. In Proc. Int. Conf. On Software Engineering, 1991.

[Grundy99] Jim Grundy, “Aspect Oriented Requirements Engineering for Component Based Software Systems”, In the Proceedings of Requirements Engineering (RE’99), June, 1999, Limerick, Ireland, IEEE Press.

[GunterMuschollPeled01] E. Gunter, A. Muscholl, and D. Peled. Compositional Message Sequence Charts. In Proc. of TACAS’01, volume 2031 of Lecture Notes in Computer Science, pages 496–511. Springer, 2001.

[Harrison et al 97] Tim Harrison and David Levine and Douglas C. Schmidt, “The Design and Performance of a Real-time CORBA Event Service,” Proceedings of OOPSLA ‘97, ACM, Atlanta, GA, October 1997.

[Henzinger98] Thomas A. Henzinger, It's about time: Real-time logics reviewed., In Davide Sangiorgi and Robert de Simone, editors, Proceedings of Ninth International Conference on Concurrency, volume 1466 of LNCS, pages 439–454. Springer-Verlag, Nice, France, 1998.

[Henzinger01] Thomas A. Henzinger, Ben. Horowitz, and Christoph M. Kirsch, “Giotto: A Time Triggered Language for Embedded Programming”, In the Proceedings of the First International Workshop on Embedded Software (EMSOFT’01), Lake Tahoe, CA, USA, October 2001.

[Hoare85] Communicating Sequential Processes, Prentice Hall, 1985.

[Huang et al 97] J. Huang, R. Jha, W. Heimerdinger, M. Muhammad, S. Lauzac, B. Kannikeswaran, K. Schwan, W. Zhaonad R. Bettati, “RT-ARM: A Real-Time Adaptive Resource Management system for Distributed Mission-Critical Applications,” in Workshop on Middleware for Distrbuted Real-Time Systems, RTSS-97, (San Francisco, CA), IEEE, 1997.

[IDL/OMG] OMG Website, http://www.omg.org.

[IraniShuklaGupta03] Sandy Irani, Sandeep Shukla and Rajesh Gupta. “Algorithms for Power Savings, SODA 2003.

[JainSchmidt97] P. Jain and D. C. Schmidt, “Service Configurator: A Pattern for Dynamic Configuration of Services: in Proceedings of the 3rd Conference on Object-Oriented Technologies and and Systems, USENIX, June 1997.

[Kiczales 97] G. Kiczales, "Aspect-Oriented Programming," in Proceedings of the 11th European Conference on Object-oriented Programming, June, 1997

[Koskimies et al. 98] Kai Koskimies, Tarja SystÄa, Jyrki Tuomi, and Tatu Männistö. Automated Support for Modeling OO Software. IEEE Software, pp. 87–94, January—February 1998.

[Krueger00] I. H. Krüger: Distributed System Design with Message Sequence Charts, Dissertation, Technical University of Munich, 2000, available at: http://tumb1.biblio.tu-muenchen.de/publ/diss/in/2000/krueger.html

[Lamsweerde00] Axel v. Lamsweerde, “Formal specification: a roadmap”, in Anthony Frankelstein ed., The Future of Software Engineering, ACM Press, 2000.

[LeeXiong01] E . A. Lee and Y. Xiong. System-Level Types for Component-Based Design. In First International Workshop on Embedded Software, vol.2211 of Lecture Notes in Computer Science. Springer, October 2001.

[LiaoTjiangGupta97] S. Liao, S. Thiang, and R. Gupta. An Efficient Implementation of Reactivity in Modeling Hardware in the Scenic Synthesis and Simulation Environment. In Proc. IEEE/ACM Design Automation Conf., 1997.

[Linda 93] Bjornson Robert, “Linda on Distributed Memory Multiprocessors”, PhD thesis, Yale University, 1993.

[Loyall et al 01] J. Loyall, J. Gossett, C. Gill, R. Schantz, J.Zinky, P. Pal, R. Shapiro, C. Rodrigues, M. Atighetchi and D. Karr, “Comparing and Contrasting Adaptive Middleware Support in Wide-Area and Embedded Distributed Object Applications,” in Proceedings of the 21st International conference on Distributed Computing systems (ICDCS-21), pp.625-634, IEEE, April 2001.

[Manna, Pnueli] Zohar Manna and Amir Pnueli, "The temporal Logic of reactive and concurrent systems", Springer Verlag, 1992

[Metropolis] Metropolis Project Web Page, http://www.gigascale.org/metropolis/infrastructure.html

[Microsoft Dnet 01] Microsoft, Microsoft(r) .NET My Services Specification, Microsoft Press, October 2001

[Misra, Chandy 88] J. Misra and K. M. Chandy, Parallel Program Design: A Foundation, Addison-Wesley, 1988.

[MohapatraVenkatasubramanian03] S. Mohapatra and N. Venkatasubramanian, “PARM: Power-Aware Reconfigurable Middleware”, in ICDCS-03.

[Mohapatra et al 03] S. Mohapatra, R. Cornea, N. Dutt, A. Nicolau, N. Venkatasubramanian, “Integrated Power Management for Video Streaming to Mobile Handheld Devices, ACM Multimedia 2003.

[Moml 2000] Steve Neuendorffer, Ed Lee, “MoML: An XML Modeling Markup Language”, http://buffy.eecs.berkeley.edu/IRO/Summary/00abstracts/neuendor.1.html

[Morse96] J. Morse, and S. Hargrave, “The increasing importance of Software”, Electronic Design, vol.44(1), Jan. 1996.

[Mousavi01] Mousavi, M.R., Rusello G., and Chaudron M. R. V, “ A Coordination Approach for the Design of Component Based Distributed Real-Time Systems”, submitted.

[Mousavi et al 2002] M. Mousavi, M. Chaudron, G. Russello, M. Reniers, T. Basten, A. Corsaro, S. Shukla, R. Gupta and D. Schmidt. Using aspect –GAMMA in Design and Verification of Embedded systems. In Proc. High level Design Validation and Test Workshop, 2002.

[O'Ryan et al 2000] C. O'Ryan, D. C. Schmidt, F. Kuhns, M. Spivak, J. Parsons, I. Pyarali, D. Levine, "Evaluating Policies and Mechanisms for Supporting Embedded, Real-Time Applications with CORBA 3.0" in Proceedings for the 6th IEEE Real-Time Technology and Applications Symposium, (Wash. D.C.), IEEE, May 2000.

[Omg99a] Object Management Group, Real-time CORBA Joint Revised Submission, OMG Documentorbos/99-02-12 ed., March 1999.

[Omg99b] Object Management Group, "Dynamic Scheduling, OMG document orbos/99-03-32 ed., March 1999.

[Omg2000] Object Management Group, "The Common Object Request Broker: Architecture and Specification, 2.4 ed., October 2000.

[Omg01a] Object Management Group, "The Common Object Request Broker: Architecture and Specification Revision 2.5, OMG Technical Document formal/00-11-07", October 2001.

[Omg01b] Object Management Group, "The Common Object Request Broker: Architecture and Specification, 2.6 ed., December 2001.

[Paulin97] P. Paulin, C. Liem, M. Cornero, F. Nacabal, G. Goossens, "Embedded Software in real-time signal processing systems: Application and architectural Trends", Proceedings of IEEE, vol. 85(3), 1997.

[Pyarali+02] Irfan Pyarali, Douglas C. Schmidt, and Ron Cytron, "Techniques for Enhancing Real-time CORBA Quality of Service," Submitted to the IEEE Proceedings. Available at http://www.cs.wustl.edu/~schmidt/corba-research-realtime.html. p. 419-435.

[Ramanathan TCAD2002] D. Ramanathan, S. Irani, R. Gupta, "An Analysis of System Level Power Management Algorithms and their effects on Latency", IEEE Transactions on Computer Aided Design, March 2002.

[RTCorba 2000] Object Management Group, "Dynamic Scheduling Real-Time CORBA Joint Revised Submission, OMG Document orbos/2000-08-12 ed.", August 2000.

[Saxena99] Saxena . A, Shukla. S, Weihmayer. R, Wu. P, "CORBA based Event Management System: A Case Study in Automatic Global Correlation", In the Proceedings of the International Conference on Parallel Processing Techniques and Applications (PDPTA'99), CRA Press, Las Vegas, June 1999.

[Schantz 2002] Richard E. Schantz and Douglas C. Schmidt, "Middleware for Distributed Systems: Evolving the Common Structure for Network-centric Applications," Encyclopedia of Software Engineering, Wiley and Sons, 2002.

[Schmidt et al 2000] D. C. Schmidt, M. Stal, H. Rohnert, and F. Buschmann, Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects, Vol. 2. New York: Wiley & Sons, 2000.

[SchmidtKuhns2000] D. C. Schmidt and F. Kuhns "An Overview of the Real-time CORBA Specification" IEEE Computer Magazine, Special Issue on Object-oriented Real-time Computing, vol.33, June 2000.

[Schmidt et al 2001] D. C. Schmiddt, S. Mungee, S. Flores-Gaitan, and A. Gokhale, "Software Architectures for Reducing Priority Inversion and Non-determinism in Real-time Object Request Brokers," Journal of Real-time Systesm, special issue on Real-time Computing in the Age of the Weg and the Internet, vol.21, no.2, 2001.

[Schmidt 2001] Douglas C. Schmidt, Sumedh Mungee, Sergio Flores-Gaitan, and Aniruddha Gokhale, “Software Architectures for Reducing Priority Inversion and Non-determinism in Real-time Object Request Brokers,” Journal of Real-time Systems, Kluwer, Vol. 21, No. 2, 2001.

[SelicGulleksonWard94] Bran Selic, Garth Gullekson, and Paul T. Ward: Real-Time Object-Oriented Modeling, Wiley, 1994.

[ShaRajkumarLehoczky90] L. Sha, R. Rajkumar, and J.P. Lehoczky, “Priority Inheritance Protocols: An Approach to Real-time Synchronization,” IEEE Transactions on Computers, vol. 39, September 1990

[Shenoy03] P. Shenoy and P. Radkov. “Proxy-Assisted Power-Friendly Streaming to Mobile Devices”. In MMCN, 2003.

[Shukla98] S. Shukla “Fault-Tolerance Patterns for Network Management Applications”, Invited Presentation at the Dagstuhl Seminar on Self-Stabilization, Dagstuhl, Germany, August 1998.

[Stankovik87] John A. Stankovic and Krithi Ramamritham, Tutorial on Hard Real-Time Systems, IEEE Computer Society Press, 1987.

[Szyperski98] C. Szyperski. Component software: Beyond Object Oriented Programming. Addison-Wesley, 1998.

[Thoen-Cathhoor00] Filip Thoen, and Francky Catthoor, “Modeling, Verification and Exploration of Task-Level Concurrency in Real-Time Embedded Systems”, Kluwer Academic Publishers, 2000.

[Udupa 99] Divakara K. Udupa “TMN: Telecommunications Management Network”, McGraw-Hill Professional Publishing, January 1999.

[U2 Partners] Revised submission to OMG RFPs ad/00-09-01 and ad/00-09-02:Unified Modeling Language 2.0 Proposal. Version 0.671 (draft). available at http://www.u2-partners.org/artifacts.htm, 2002.

[VenkatasubramanianTalcottAgha01] Nalini Venkatasubramanian, Carolyn Talcott, Gul Agha, "A Formal Model for Reasoning about Adaptive QoS-Enabled Middleware ", FME 2001, Germany, March 12-16, 2001.

[Venkatasubramanian et al 2001] Nalini Venkatasubramanian, Mayur Deshpande, Shivajit Mohapatra, Sebastian Gutierrez-Nolasco and Jehan Wickramasuriya, “Design & Implementation of a Composable Reflective Middleware Framework", ICDCS-21, April 2001.

[Wang et al 01] Nanbor Wang, Douglas C. Schmidt, Kirthika Parameswaran, and Michael Kircher, “Towards a Reflective Middleware Framework for QoS-enabled CORBA Component Model Applications,” IEEE Distributed Systems Online special issue on Reflective Middleware, 2001.

[WhittleSchumann00] J. Whittle and J. Schumann. Generating Statechart Designs From Scenarios. In International Conference on Software Engineering (ICSE 2000), 2000.

[XiongLee2000] Y. Xiong and E. A. Lee. An Extensible Type System for Component-Based Design. In the 6th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, vol. 1785 of Lecture Notes in Computer Science. Sringer, april 2000.

[Yuan et al 2003] W. Yuan, K. Nahrstedt, S. Adve, D. Jones, and R. Kravets. “Design and Evaluation of a Cross-Layer Adaptation Framework for Mobile Multimedia Systems”. In MMCN-03.

[ZinkyBakkenSchantz97] J. A. Zinky, D. E. Bakken and R. Schantz, “Architectural Support for Quality of Service for CORBA Objects,” Theory and Practice of Objects Systems, vol. 3, no.1, pp.1-20, 1997.

Automatic detection of service interactions from graphical specifications

Hélène Jouve, Pascale Le Gall, and Sophie Coudert

L.a.M.I., CNRS UMR 8042
Université d'Évry
523 places des Terrasses
91000 Évry, France
Tel: (+33) 1 60 87 39 14 Fax: (+33) 1 60 87 37 89
{hjouve, legall, coudert}@lami.univ-evry.fr

Abstract. The paper presents a systematic method for detecting interactions from telecommunication service specifications. Services are graphically specified by means of formalised diagrams. The detection of service interactions is based on static analysis of the diagrams and reveals two kinds of interactions. Direct interactions occur when a message triggers two services. Indirect interactions occur when the triggering of a service simulates a triggering message of another one. The method allows to compute interactions in terms of subscription configuration, triggering message and triggering condition. It is fully automatic without any need of additional knowledge, implemented in Prolog and illustrated in the paper on a small running example.

Keywords : Telecommunication services, service interaction, static analysis, subscription configuration, unification.

1 Introduction

Telecommunication systems are constantly evolving due to the introduction of new services (also often called features) which increase the set of available functionalities. [5], [7], [10], [3] give a general presentation of the feature interaction problem which is still pressing and largely unsolved. A service may be understood as a functional and optional unit modifying an underlying basic call service[1]. Network operators are strongly interested in decreasing the life cycle of the service design. Most of the time, services are designed separately. Thus, their integration may lead to unexpected interactions, when the mutual influence between two or more services makes the system behave differently from the expected behaviour of each involved service. These interactions may be considered as acceptable or unpleasant. In order to propose a care-free service set, network operators should prevent the emergence of undesirable interactions. The first step to tackle the service interaction problem is clearly the detection: the knowledge of interactions is necessary to integrate services in a way that satisfies the need of users

[1] As in many other articles dealing with services, this basic call service pre-existent to all further services is simply called POTS for Plain Old Telephone Service, ensuring simple calls between users.

subscribing these services. Indeed, roughly speaking, a pragmatic service-oriented design method ([4]) includes the following steps: interaction detection, expert judgement to qualify interactions as desired or not, integration mechanisms to keep the desired interactions while discarding the undesired ones.

Since properties of the integrated system obviously inherit from properties of individual services, formal methods are promising in detecting interactions. Verification techniques such as theorem-proving, model-checking or formal testing have been proposed for that purpose ([10], [18], [1], [6] [2], [14], [8], [4], [23]). The main common drawback of most of these approaches is that their application needs costly efforts such as the construction of the whole model of the telecommunication system as an automaton, the systematic exploration of the built automaton, . . . which can reveal intrinsic complexity limitations. However there are also some works ([21], [12], [9] . . .) based on static analysis methods. They generally deal with prepost formulas. Roughly speaking, they study how properties expressed by means of preposts formulas issued from different services can be in conflict. Either they lead to irreconcilable situations from compatible preconditions (in other words, this is a non-determinism case) or they lead to situations where some preposts of one of the two services cannot be applied any more. These methods are essentially based on heuristics.

The work we present here takes place in the family of static methods for detecting interactions. It addresses high-level intrinsic interactions since services will be described from the user's point of view, only capturing the observable behaviours. The result is an actually implemented algorithm for detecting interactions from formalised diagram specifications. Indeed, a classical way to achieve such high level specifications is to synthesise behaviours under the form of representative message sequences ([22]). In our approach, we consider sequences which are made of messages sent between the network and the users. This point of view discards as much as possible details about the network internals. We use a common specification style based on diagrams of essential communication scenarios, as it has been done in the two recent service interaction contests ([11], [15]) which defined services by means of diagrams known as respectively Chisel sequence diagrams and state transition diagrams. Using such intuitive diagrams, we propose a systematic method for detecting logical interactions based on static analysis. To facilitate the definition of algorithms, we have been induced to define our own framework of diagram specification. In particular, we precisely define variable scope, variable substitution or the mechanisms of aliases introduced to model loops. The formalisation allows us to model service diagrams as simple trees, and thus as Prolog terms to be analysed. Indeed, we have fully implemented our method using logic programming in order to take advantage of its symbolic facilities. With respect to the prepost approaches, dealing with diagrams for static analysis offers a better consideration of state successions and a finer characterisation of state equivalence by providing a more complete characterisation of the future of a state. In order to assess the value of our method, we have first analysed the computed results with respect to our understanding of the service specifications. Then, we have compared the results provided by our approach with the results of the FIW98 and FIW00 contests ([5], [7]). The details of this study can be found in [13]. Our results are globally significant since we almost find all already detected interactions. To our opinion, the slight differences which remain may be explained by the fact that some service description in the contests were ambiguous, and thus may be specified in different ways. Thanks to our static analysis, problematic configurations (states and subscriptions which lead to interactions) are infered and not proposed *a priori*. On the contrary, in most of verification approaches based on test or model-checking technics, a number of phones and a subscription configuration have to be provided in order to build the model to check. Our method allows to automatically elicit knowledge about minimal subscription configuration needed to reveal interactions. Moreover, by considering intentional specified behaviour of services, we also go beyond obvious syntactical criteria which for example, lead to cases of non-determinism. We find both obvious interactions raised by a triggering event common to different services and indirect interactions occurring when the effect induced by a first service meets the triggering condition of the second one. Each computed interaction is provided with the minimal informations of the subscription configuration, state conditions over the phones involved in the subscription configuration, and the triggering message at the source of the interaction.

The rest of the paper is structured as follows. Section 2 introduces our characterisation of interactions and the running small example used to illustrate our approach. In Section 3, we present some of the service diagram specifications we deal with. In Section 4 we explain our method and show on a classical example how it allows to compute interactions. By lack of space, the algorithm will be given only at the level of its main steps : technical details or minor steps will be skipped. The algorithm description will combine intuitive and technical considerations.

2 Characterisation of interactions

2.1 A simple intuition

Let us introduce the interaction problem from an classical example involving two services : TCS (Terminating Call Screening) and CFB (Call Forward when Busy). TCS allows the subscriber to prevent incoming calls from a specified phone. When such calls are performed, they are rejected and then, the callers listen to a specific refusal recorded message, denoted by *TCSmsg* in the sequel. CFB ensures that all calls towards the subscribing phone are forwarded to another phone (specified by the subscriber), as soon as the subscriber is busy.

Direct interaction: If a phone B is forwarding incoming calls to a phone C and rejecting incoming calls from a phone A, a interaction occurs when A calls B (message $A.\mathit{call}(B)$) while B is busy. With respect to CFB, the message $A.\mathit{call}(B)$ should aim to connect A and C, while respecting TCS, it should originate the sending to A of the *TCSmsg* recorded message. The network could so react to the triggering message $A.\mathit{call}(B)$ in two different ways. This double incompatible answer precisely characterises direct interactions. Such interactions are often seen as a case of non-determinism because in a same network state, the sending of a message requires two different responses from the network.

Indirect interaction: A more subtle interaction occurs when a contradiction indirectly arises between service requirements. It is the case when the expected behaviour specified for a service meets the triggering conditions of another service. Such cases are said to be semantical interactions by [21]. Let us clarify this class of interactions by considering a phone B subscribing the CFB service with C as target and C subscribing the TCS service with A in its screening list. Then, an interaction occurs when A calls B which is already in a busy state. Indeed, the CFB service asks for the network to react as if it had received the intentional message $A.\mathit{call}(C)$ instead of the initial message $A.\mathit{call}(B)$. An interaction occurs since the simulated message could originate the refusal message *TCSmsg* to A while the CFB service requires to put in connection A and C. This situation is an indirect case of non-determinism and so, should be also considered as an interaction. We will qualify such interactions as indirect. In the sequel, the simulated messages will be said to be *intentional*. Thus, through the intentional messages associated to the service invocations, interactions may be indirectly perceived.

Interactions may be characterised by information of different nature. The first information, called call configuration in the sequel, is the knowledge of the involved phones provided with their subscriptions to the services. In the first example (direct interaction), the configuration is given by three phones, respectively A, B and C such that B subscribes to TCS in order to prevent incoming calls from A and to CFB with C as target. In the second example (indirect interaction), the configuration is given by three phones too, but with another subscription arrangement: B forwards incoming calls to C which prevents calls from A. The second information is the triggering message causing the interaction: $A.\mathit{call}(B)$ in both examples. The last useful information implied in interactions is the triggering condition, which gives pertinent knowledge on the phone states to reveal the interaction. In both examples, B has to be busy just as A tries to call it. It expresses the minimal condition on the system state for the services to be jointly exercised so that an interaction occurs. In the sequel, another aspect will be taken into account to characterise interactions. According to the fact that the responses specified by the two services are the same or not, the corresponding interactions will be qualified as visible or invisible. Thus, in the previous examples, the two described interactions are visible. The following Section is devoted to detail the elements defining interactions (as indirect or not, visible or not).

2.2 Interactions

Service specifications contain elements of different nature such as users which are assimilated to the network phones, recorded messages like the *TCSmsg* message, tones like *dialtone*, *ringingtone*, *ringbacktone*, *linebusytone* or *disctone* indicating to the phone user in which state is its phone, etc.

Definition 1. *Let us consider S a set of sorts including at least the sorts recorded, phone, tone. Each sort, except the sort phone, is provided with a set of constructors, generally simple constants like dialtone for the sort tone or TCSmsg for the sort recorded.*

For any service name $F : s_1 \times \ldots s_n$ with $\forall i \in 1..n, s_i \in S$ and $n \geq 0$, a **subscription** *to the service F is of the form User : $F(t_1, \ldots, t_n)$ with, for all i in $1..n$ t_i a term of sort s_i. User is called the subscribing user while for each $s_i = phone$, User$_i$ is called an argument user.*

The set of sorts is useful to declare all the basic data types used in a service specification. With respect to the sort *phone*, in the concern of generality, we will represent arbitrary users by simple anonymous variables. Thus, in the sequel, V will denote the set of phone variables representing phone users, and provided with a total order relation $<$ when technical reasons ask for it. These variables will be denoted as A, B, C, $\ldots$, or $User_i$, and possibly indexed by a service name.

Provided that the Terminating Call Screening service is introduced by the notation TCS : *phone*, an example of subscription is given by $A : TCS(B)$ and indicates that A subscribes the service *TCS* with B as forbidden origin of incoming calls[2]. Moreover, a user variable A without any particular subscription subscribes by default the basic call system, named as *POTS*. Such a situation is denoted by $A : POTS$. In practice, most of services are defined from the knowledge of the underlying common basic call system *POTS*. It can be generalised by introducing a partial order relation over services. Since *POTS* is a basic service, all services F depend on the *POTS* description, but some other dependences between services can also exist. Thus, the notation $F_1 < F_2$ means that the specification of the service F_2 is based on the knowledge of the one of F_1. Such a facility will allow us to express a more complex service on some intermediate services in order to reduce the specification size of F_2 by referencing elements of F_1 within the specification of F_2.

Definition 2. *A **call configuration** is the given of a set of subscriptions such that each user variable occurring as argument user of a subscription is also a subscribing user of another subscription of the configuration.*

*A call configuration is said to be **non-degenerated** if any subscription does not contain several occurrences of a same variable*[3].

*Let $F_1, \ldots F_n$ be n service names. A $(F_1, \ldots, F_n)$-**call configuration** is a non-degenerated call configuration containing at least a subscription to F_i for each i in $1..n$.*

For example a (TCS, CFB)-call configuration for the first given interaction is $\{A : POTS, B : TCS(A), B : CFB(C), C : POTS\}$ while $\{A : POTS, B : CFB(C), C : TCS(A)\}$ concerns the second one. Both are non-degenerated, exactly contain a subscription for each service of (TCS, CFB). Let us remark that to ensure that any user variable subscribes to at least a service, some user variables subscribe to *POTS*, the least service name by hypothesis.

Let us remark that (F_1, F_2)-call configurations are similar to possibly interacting configurations as defined in [19] in order to enumerate all call configurations likely to bring about some interactions between the services F_1 and F_2. [19] *a priori* builds all the possibly interesting configurations in which one has to search for possible interactions. But the number of such configurations remains large. Contrary to [19], our purpose is not to *a priori* build them, but to detect interactions and to be capable for

[2] If one wants to express that A prevents incoming calls from both B and C, then A has to subscribe twice the TCS service: $A : TCS(B)$ and $A : TCS(C)$.

[3] Thus, for example, a call configuration cannot contain a subscription neither of the form $User : F(User)$ nor of the form $User_1 : G(User_2, User_2)$.

each of them of infering the knowledge of an underlying call configuration representative of the interaction. To firstly consider simple cases and like some other works ([20]), we restrict ourselves to $(F_1, \ldots, F_n)$-service configurations for $n = 2$ in order to focus on the detection of interactions occurring between only 2 service subscriptions.

We have already outlined that interactions will be also characterised by triggering information, decomposed into two parts; the triggering message and the triggering conditions:

Definition 3. *Let us consider P a set of* ***predicates*** *provided with an arity on*[4] S^+. *A* ***system state*** *is a set of literals either of the form $p(t_1, \ldots, t_n)$ or of the form*[5] $\sim p(t_1, \ldots, t_n)$ *with p a predicate of P provided with the arity $s_1 \times \ldots \times s_n$ and t_i for $i \in 1..n$ a term of sort s_i.*

Let us consider PM (resp. NM) a set of phone (resp. network) message names provided with an arity on S^*. For a phone message name m in PM of arity $s_1 \times \ldots \times s_n$, the ***phone message*** *$A.m(t_1, \ldots, t_n)$ with t_i of sort s_i means that the phone A sends the message $m(t_1, \ldots, t_n)$ to the network. For a network message name m in NM of arity $s_1 \times \ldots s_n$, the* ***network message*** *$m(t_1, \ldots, t_n).A$ with t_i of sort s_i means that the network sends the message $m(t_1, \ldots, t_n)$ to the phone A.*

All symbols needed to specify a service (as S, P, PM, NM for sorts, predicates, messages and so on) are grouped in a signature, denoted by Θ in a generic way.

Definition 4. *A* ***triggering message*** *is a phone (or possibly a network) message while a* ***triggering condition*** *is a finite set of literals.*

For example, if B subscribes to the *CFB* service with $B : \mathit{CFB}(C)$, then $A.\mathit{call}(B)$ is a possible pertinent triggering message and $\{\sim \mathit{idle}(B), \mathit{dialing}(A)\}$ is the associated triggering condition with *idle* and *dialing* predicates of arity $phone$ interpreted according to their obvious meaning.

Definition 5. *Let (F_1, F_2) be two services names.*

*A (F_1, F_2)-**service interaction** $I = (CC, TM, TC)$ is the given of a (F_1, F_2)-call configuration, a triggering message and a triggering condition such that the user variables occurring in TM or TC are variables of the configuration CC.*

For example, $(\{A : \mathit{POTS}, B : \mathit{TCS}(A), B : \mathit{CFB}(C), C : \mathit{POTS}\}, A.\mathit{call}(B), \{\sim \mathit{idle}(B), \mathit{dialing}(A)\})$ and $(\{A : \mathit{POTS}, B : \mathit{CFB}(C), C : \mathit{TCS}(A)\}, A.\mathit{call}(B), \{\sim \mathit{idle}(B), \mathit{dialing}(A)\})$ are two service interactions for $(\mathit{TCS}, \mathit{CFB})$. Our purpose is to automatically elicit such interactions from the static analysis of the service specifications. Indeed, [17] outlines that observing and analysing statically graphical descriptions of services allow to find most of the interactions. We want to emphasise this point of view in giving more importance to the role of variables and intentional messages. Our method is extending the one developed in [21] to specification given under the form of diagrams instead of state transition rule set. Indeed, [21] already elicits pertinent knowledge, such as inhibited primitive and intention primitive in order to explain the presence of interactions. Our approach may also be compared to the one described in [9] since

[4] S^+ denotes all the non empty words on S while S^* denotes all the words on S.

[5] The symbol $\sim$ is here used as negation symbol.

they also extract interactions by static analysis from graphical specifications, except that they do not deal with indirect interactions.

3 Diagrams

In order to ease static analysis of our specifications, we opt to denote them as rooted trees which can be represented as finite terms. However, some nodes are aliases pointing at a marked node to express loop schemas. Our specifications are composed of a service call configuration, a diagram and a constraint set. The diagram describes the system behavior by making explicit all representative allowed message exchanges between the network and the phones. The call configuration expresses a kind of general precondition in relation to the diagram variables: the variables of the diagram are precisely the ones occurring in the subscription terms of the call configuration. The constraints allow to express semantic links between state predicates which restrict the set of admissible states of the system.

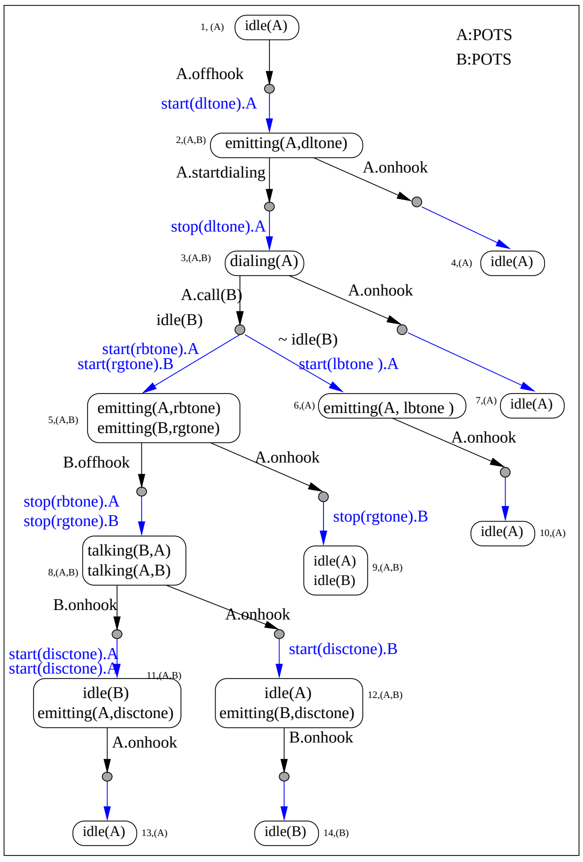


Fig. 1: *POTS* diagram

talking$(A, B) \Rightarrow\sim$ *idle*(A) is, for example, a constraint imposing that a phone A cannot be at the same time in communication with another one and in an idle state.

Each diagram admits a graphical representation. To give a general idea of our service specification, let us first introduce the one of the *POTS*, the basic communication service.

POTS has for call configuration a double subscription of the *POTS* service by the variables A and B. The diagram begins with the root node specifying that A is in an idle state. Each node expresses properties on phones that are required both before the messages labelling the outgoing edges are sent and after the messages labelling the entering edges are sent. Most of the edges are labelled by a phone message and an answer of the network. At each diagram node, each potential future event (phone message) has to be specified so that each possible transition figures on the diagram. The network can give two different answers to the output $A.\mathit{call}(B)$ which depend on whether B is *idle* or not. This illustrates the interest of using conditions on edges. The common part of the two edges, i.e. the $A.\mathit{call}(B)$ label, is then shared: it makes the diagram more legible. Let us remark that on this diagram, the condition is empty for all the other edges.

Let us also remark that each node is annotated by a number n and a list of variables, the ones which are precisely in the scope of the considered node. The variable list (A, B) of the *POTS* service is ordered according to $A < B$. We do not detail all uses of symbols involved in the *POTS* diagram which in fact, naturally correspond to their usual meaning in the field of telecommunication services. For example, $\mathit{start}(\mathit{disctone}).A$ is a network message in direction of A under the form of a specific tone indicating to A that there is no longer a user talking to it. In the same way, *ringbacktone* holds for the ringbacktone, *ringingtone* for the ringingtone, *dialtone* for the dialing tone ...

Our diagrams are very close to the Chisel ones ([5]). They specify an alternation of messages from phones to the network and from the network to phones. They also share the notion of system states with state transitions diagrams used in [16]. As previously explained, we systematically use variables to model phones. Indeed, substitution mechanisms will assign different rôles to the occurring phones, particularly, in different subscription terms of a call configuration.

3.1 Definitions

Definition 6. *Let us consider two services provides with F as service name. Let N_F be an arbitrary set. A **node identifier** for F over N_F is a couple (F, n), where n belongs to N_F. An **alias** for F is a couple $(ni, (v_1, \ldots, v_k))$ where ni is a node identifier for a service G such that $G \leq F$ and v_i for $i \in 1..k$ is a variable of V. The alias set for F is denoted by $\mathit{Alias}_\Theta(V, F)$.*

In practice, the considered set N_F will be a finite subset of the set of all natural numbers so that each node of the diagram of a service specification will simply be denoted by an arbitrary number and implicitly marked by the service name. Obviously, this systematic node marking will serve us to define alias mechanisms.

The node identifier appearing in an alias for F has to refer to a node of a service preceding it according to the preorder relation defined over the services. k is *a priori* the number of variables under the scope of the node indicated by the alias.

Definition 7. *Let us consider a service of name F and provided with a signature Θ. A **node label** over Θ is either a system state over Θ or an alias for F.*

$\mathit{NodeLabel}_\Theta(V)$ denotes the set of node labels made of system states over Θ.

A system state for a node label partly characterizes the current state of the system while an alias will serve us as a link to a referenced node. It allows us both to be sparing of specification redundancies and to model intrinsic loop phenomena within specifications.

Definition 8. *Let Θ be a signature for the service name F. An* ***edge label*** *over Θ is either a 3-uple made of a phone message, a literal set and a set of network messages or a couple made of a literal set and a set of network messages.*

The literal set is called condition *while the set of network messages is simply called an* answer. *The set of all edge labels for F is denoted by $EdgeLabel_{\Theta}(V)$.*

An edge label expresses input and output messages leading to a system state evolution. The outputs (the network answers) are sent to the phones. They can occur either in direct reaction to a network input (a phone message) and/or according to some conditions expressed on the system states. The conditions on edge labels are necessary to the given input/output message exchange but not required in the source node of the transition.

Definition 9. *Let Θ be a signature for the service name F provided with its set of node identifiers built from the set N_F.*

A ***diagram*** *D for F, over Θ is a 4-uple $(N_F, Edge, node, edge)$ where $(N_F, Edge)$ is a rooted tree* [6] *and node and edge are the labelling functions respectively from N_F to $NodeLabel_{\Theta}(V) \cup Alias_{\Theta}(V, F)$ and from Edge to $EdgeLabel_{\Theta}(V)$ such that any node labelled by an alias is a leaf of the rooted tree.*

In a diagram, nodes correspond to the state descriptions and edges to the transition descriptions. As diagrams are trees, we inherit of the notion of *subdiagrams* defined as *subtrees* of the global tree. These subdiagrams may be reduced to simple alias nodes. Moreover, for each node, we can consider *branches* following it, as defined by any couple composed of an outgoing edge stemming from the node and the subdiagram it points to. Aliases are in fact references to pre-existing (sub)diagrams: an alias $((G, j), (v_1, \ldots, v_k))$ of a F diagram designates the node referenced by (G, j) in the G diagram provided that of course, $G \leq F$. If G is equal to F itself, then the F diagram contains a loop. For example, the diagram associated to the *Call Waiting* service contains a communication loop since the subscriber can switch the held phone with the one it's talking to. More precisely, the alias $((G, j), (v_1, \ldots, v_k))$ points at the subdiagram of the G diagram whose the root node is (G, j). The variables occurring in the G subdiagram are substituted by $(v_1, \ldots, v_k)$, according to the order defined over the G variables : in other words, the least variable in the scope of the subdiagram defined by the root (G, j) is replaced by the variable v_1. It is thus required that k coincides with the number of variables in the scope of the (G, j) node.

Let us now introduce the CFB diagram which is based on the *POTS* one. The beginning of the service diagrams coincides with an intermediate node of the *POTS* diagram. It ensures that the initial states of the CFB service is reachable from a network only involved with users subscribing the basic call system *POTS*. The CFB diagram con-

[6] A directed graph is defined by a couple $(Node, Edge)$ where *Node* and *Edge* respectively define the set of all nodes and the set of all edges (defined as couples of their source node and target node). A rooted tree is a connected directed graph with an identified node, the root, considered as initial and such that there is no cycle in the graph, any node being reachable from the root by a consecutive edge sequence.

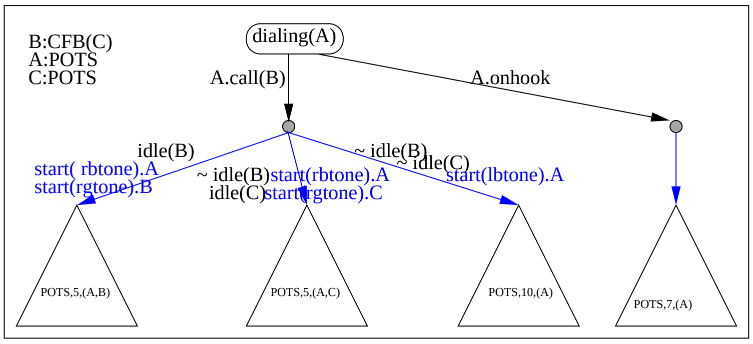


Fig. 2: CFB diagram

sists in adding both edges for new messages A.*call*(B) and conditions for A.*call*(B) edges: the conditions depend on the state of the two phones B and C and according to the cases, the service is triggered or not. In particular, the first branch stemming from the *dialing*(A) node is similar to the corresponding *POTS* branch while the second and the third ones are new. Indeed, they are concerned with the variable C occurring as an argument of the CFB service. There are two subdiagrams equivalent, up to some variable substitution, to a *POTS* subdiagram, precisely the one whom node identifier is $($*POTS*$, 5)$. In order to explicit these similarities, we prefer to use aliases instead of detailing these diagrams. Alias nodes are graphically represented by means of triangles instead of ovals used for state nodes.

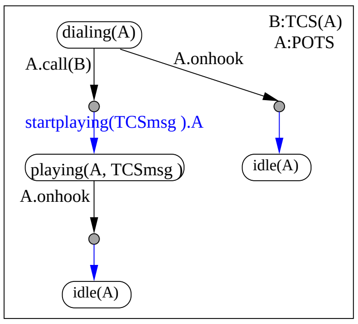


Fig. 3: TCS diagram

With the subscription information that a phone B subscribes the TCS service with a phone A as parameter, the TCS service simply amounts to introduce a new answer from the network to the message A.*call*(B). The network sends to the phone A a specific recorded message. It explains why the specification diagram is simple and concise. For simplicity, the node identifiers are left implicit. But a diagram does not suffice by itself to specify a service. Some informations lack about state predicates. One has to specify that a phone A cannot satisfy in the same state the predicates *playing*$(A,$ *TCSmsg*$)$ and $idle(A)$. Such requirements are specified by means of axioms, called constraints, which are added to a diagram.

Definition 10. *Let Θ be a service signature. A* ***constraint*** *is a formula $l \Rightarrow l'$ where l and l' are both literals over Θ.*

Constraints play the rôle of invariants on the set of reachable system states. We do not develop this point but they are particularly useful to define system states and conditions in a minimal way. For example, *talking*$(X, Y) \Rightarrow\sim$ *dialing*(Y) is a constraint for *POTS* specifying that a given phone Y cannot be at the same time in a *dialing* and *talking* state. Obviously, as we have said it for the TCS service, each service adds its appropriate constraints, mainly by introducing new predicate symbols. These constraints have to be explicitly specified and joined to the service diagram : a service specification is composed of a signature, a diagram provided with subscription informations, and constraints. However, the constraints are rather obvious to write.

4 The detection method and the corresponding algorithms

4.1 The main steps of the method

In fact, the proposed method combines two major steps corresponding to two different algorithms which are more precisely presented in the two following subsections. The first algorithm extracts the triggering informations of a service from its diagram specification. The idea is to simply identify the points in the diagram where the specification differs from the one of the reference system (the point where the service is supposed to be plugged, generally a point included in the *POTS* diagram). While detecting these divergences, the algorithm collects the configuration informations leading to them as term of subscriptions, states and triggering message. Moreover, the diagram is annotated with the appropriate intentional messages. The second algorithm exploits information computed by the first one by comparing the whole annotated subdiagram stemming from the activation point of the first service to the triggering conditions of the second one. Like the previous algorithm, while searching, this one collects the informations relative to the configuration which leads to a conflict and then provides them as a result. This algorithm can thus be qualified as an interaction elicitation algorithm.

4.2 Extraction of the triggering informations

Principle of the algorithm To compute the triggering information of any service F, we compare its diagram to the default service diagram, i.e. generally the POTS one. Below, we outline the main steps to extract all pertinent informations related to service triggering. Notice that we use a special kind of unification: the computed unifiers are constrained by preventing each variable of a diagram from being bound to more than one variable of the other diagram. This prevents the confusions between the different phones occurring in a given specification. Indeed, in the intention of the specifier, each variable of phone of a service diagram corresponds to a specific rôle in the described set of scenari. Confusing them is not relevant: this differs from the prepost specifications, for example, where the far past and future of states are not explicitly specified. Considering this precision about unification mechanism, the principle of the algorithm is the following one:

• If the initial nodes of the POTS and F services match, the comparison produces a unifier denoted by λ. If the two initial nodes do not match (see the *CFB* diagram for example), we look for a POTS node coinciding to the initial node of the service in order to proceed with the corresponding *POTS* subdiagram. If finding this analogous node in the POTS diagram is not possible, we consider that the initial node of the F diagram is in fact a specification error.

• Once a POTS node has be found as a mirror node of the F initial node, the comparison of the two diagrams is pursued with the comparison of the subdiagrams stemming from these two similar nodes. Diagram paths are recursively compared two by two, one outgoing from the F initial node, and the other one outgoing from the similar POTS node. This provides us with a set of path peers which exactly match in the context of the unifier λ (recursively extended to the new encountered variables while descending the compared paths) and a set of paths outgoing of the F initial node which do not match any of the POTS paths. The remarkable set is the one of mismatching branches provided with their corresponding unifier: they are built from mismatching paths such that the branch edge is the first edge of the path without any equivalent node in POTS. The edge source is the activation point, the edge user message is the triggering message, the union of the edge condition and the node label of the source node is the triggering condition and finally, the subdiagram linked to the edge target is simply called the activation subdiagram. Activation branches are grouped together as soon as they share both their activation point and triggering message.

• Once activation branches has been computed, the static analysis is pursued in order to collect all pertinent knowledge about these activations. First step: Elicitation of intentional messages associated to the activation branch, if they exist. It consists in looking for a network answer provided with its POTS subdiagram similar to those of the activation branch, up to some restricted unifications. If the search is successful then the user message at the origin of the corresponding POTS subdiagram is precisely the intentional message we are looking for. Of course, this search is led taking into account alias mechanisms. At each intentional message is associated an intentional condition: the union of the condition labelling the corresponding POTS edge and the label node of its node source. Second step: Elicitation of intentional messages located in subdiagrams of the activation diagram. Indeed, such intentional messages are likely to interact with triggering messages of another service, and thus, may be at the origin of an indirect interaction.

To conclude, triggering informations are basically made of the given of a set of data structured as follows: call configuration, triggering message, triggering condition, annotated activation diagram and intentional messages (associated to the triggering message), which is illustrated by the two following examples. In the sequel, by lack of place, the annotated activation diagram will be left implicit: they are given under the form of a network message and the corresponding annotated activation diagram.

Triggering information of *CFB* The informations computed by the algorithm for the *CFB* service are synthesized in the following tabular:

Call configuration	Triggering message phone msg.	Condition	Intentional message	Condition
A : *POTS* B : *CFB*(C) C : *POTS*	$A.call(B)$	$\sim idle(B)$ $dialing(A)$ $idle(C)$	$A.call(C)$	$idle(C)$ $dialing(A)$
A : *POTS* B : *CFB*(C) C : *POTS*	$A.call(B)$	$\sim idle(B)$ $dialing(A)$ $\sim idle(C)$	$A.call(C)$	$\sim idle(C)$ $dialing(A)$

These results are the expected ones: by confronting the *CFB* diagram with the POTS diagram, we can see that the activation point of *CFB* is clearly the $dialing(A)$ node. From this node, the *CFB* diagram contains two branches both sharing the $A.call(B)$ message and the $\sim idle(B)$, $dialing(A)$ condition but respectively adding the $idle(C)$ and $\sim idle(C)$ conditions. Thus, $A.call(B)$ the triggering message. The corresponding activation diagrams take place in the lacking Activation column of the tabular, together with the respective network answers ($\{start\ (ringingtone).C,\ start(ringbacktone).A\}$ and $\{start(linebusytone).A\}$). And as expected, the computed intentional messages are in the first case $A.call(C)$ (with the $idle(C)$ condition) and in the second case $A.call(C)$ (with the $\sim idle(C)$ condition).

Triggering information of *TCS* In the *TCS* diagram, the two *POTS* branches existing for the $A.call(B)$ message are suppressed and replaced by a new branch having a different associated activation diagram : the network sends the playing of a recorded refusal message. This message is really specific to *TCS* so that any intentional message can be found. So the *TCS* triggering information the programs returns is simplified:

Call configuration	Triggering message phone msg.	Condition	Intentional message	Condition
A : *POTS* B : *TCS*(A)	$A.call(B)$	*dialing*(A)	none	none

4.3 Elicitation of interactions

Principle of the algorithm In order to detect interactions between two services F_1 and F_2 (possibly $F_1 = F_2$), we apply a static analysis on their respective triggering informations. Our interaction search is decomposed in main steps according the kind of interactions we are looking for. At first glance, there are three principal cases explaining that a activation of F_1 causes an interaction with F_2.

- F_1 and F_2 share the same triggering message (direct interaction).
- The intentional message associated to the triggering message of F_1 coincides with the triggering message of F_2.
- The activation of F_1 is likely to activate F_2 via real or intentional messages.

The first step is based on a comparison of the triggering messages and thus, concerns direct interactions. The second step concerns intentional messages compared with triggering messages of the other service. The third step amounts to apply the first and second step at each edge of the activation diagram. All steps are based on a same comparison technic and according to the fact that messages are triggering or intentional ones, the interpretation of the detected interactions differs. Moreover, we can also deduce informations about the visibility of the interactions. This is shortly detailed in the four following points:

• **The comparison principle:** For $i = 1, 2$, let CC_i, TM_i, C_i be respectively a call configuration, a triggering message and a triggering condition associated to the F_i specification. Moreover, let us denote IM_i and IC_i the intentional message and associated condition when they exist. The comparison of (triggering or intentional) messages is considered as successful if they are unified up to a restricted unifier λ (which do not unify variables of a given diagram) and provided that their associated conditions are compatible with respect to the constraints and λ. In other words, it means that there exists at least one state satisfying both $\lambda(C_1) \cup \lambda(C_2)$ and the constraints of the specifications. As soon as the comparison of messages provides us with a unifier such that the conditions are compatible, there is clearly an interaction. Then, the corresponding call configuration CC results from the union of the two service call configurations CC_1 and CC_2, once the unifier has been applied and each useless POTS subscription has been suppressed of the global subscription. More precisely, if the resulting configuration contains for example the subscriptions A : *TCS* and A : *POTS*, then the POTS subscription is deleted since the POTS subscription is considered as a default one.

• **Direct interactions (with two triggering messages):** The detected interaction (CC, TM, TC) is given by $TM = \lambda(TM_1)$ (also equals to $\lambda(TM_2)$ by hypothesis) and $TC = \lambda(C_1) \cup \lambda(C_2)$, the call configuration CC being already defined. Thus, for each possible unifier λ, there is an associated direct interaction.

• **Indirect Interactions (with a triggering message and an intentional message at the activation point):** Let us suppose that the intentional message is the F_1 one, the messages IM_1 and TM_2 are unified up to an unifier λ and the conditions IC_1 and C_2 are compatible. Then, it means that the intentional message IM_1 corresponds to an F_2 triggering message. In other words, when F_1 is triggered, the expected behaviour meets the triggering condition of a F_2 triggering. For each possible unifier λ, there is an indirect 2-service interaction denoted by (CC, TM, TC) where TM is the message TM_1 corresponding to the intentional message IM_1, up to the unifier λ, $TC = \lambda(C_1) \cup \lambda(C_2)$. Such an interaction can be understood as: "in the configuration CC, when F_1 is triggered on the message TM_1, all happens as if the network had received the message IM_1, at the origin of an indirect F_2 triggering".

• **Indirect Interactions (with messages inside the activation diagram):** the whole activation diagram is covered by applying the two previous comparison mechanisms at each encountered edge.

• **Visibility of interactions :** As the triggering informations contain also activation diagrams, we are able to determine whether the detected interaction is sensible or not. If after a joint triggering of F_1 and F_2 as given in an interaction, the two activation diagrams are similar up to the appropriate unifier, then the interaction is perceived as *invisible*. In all other cases, it is a *visible* one. This means that there really exists a difference between the service requirements which is observable by an user. Of course, nothing is said about its seriousness.

We will now illustrate how our method can be applied on the *CFB* and *TCS* services before giving the corresponding result of the program. The research begins by the **interactions on triggering messages**. *CFB* and *TCS* have the same triggering messages : $A_{CFB}.\mathit{call}(B_{CFB})$ and $A_{TCS}.\mathit{call}(B_{TCS})$ on which the restricted unification provides the unifier $\lambda = [A_{CFB} = A_{TCS}, B_{CFB} = B_{TCS}]$ leaving C_{CFB} unbound. The activation conditions are compatible : $\{\sim \mathit{idle}(B_{CFB})\} \cup \emptyset$. At this step, we can affirm *TCS* and *CFB* *interact* because they are triggered at the same time. Let us detail the interactions.

The A_{CFB} call will be forwarded to C_{CFB}: if C_{CFB} is *idle*, A_{CFB} will perhaps be connected to it, and in the other case ($\sim \mathit{idle}(C_{CFB})$), A_{CFB} will receive the *linebusytone* tone. By the *TCS* triggering, A_{CFB} should receive the *TCSmsg* recorded message in both cases. The two expected behaviours do not match. So, a simultaneous triggering of *CFB* and *TCS* is perceived by the user as a requirement conflict. It is a typical example of a *visible* interaction. We can now apply λ to compute the two resulting interactions composed of the same call configuration $CC = \{A : \mathit{POTS}, B : \mathit{CFB}(C), B : \mathit{TCS}(A), C : \mathit{POTS}\}$, the same message $A.\mathit{call}(B)$ and two different conditions, respectively $\{\sim \mathit{idle}(B), \mathit{idle}(C), \mathit{dialing}(A)\}$) and $\{\sim \mathit{idle}(B), \sim \mathit{idle}(C), \mathit{dialing}(A)\}$). Let us now consider **intentional messages**. There is no intentional message for *TCS*. The first activation branch of *CFB* (holding for $\mathit{idle}(C)$) has an intentional corresponding message: $A_{CFB}.\mathit{call}(C_{CFB})$. It can be unified with $A_{TCS}.\mathit{call}(B_{TCS})$ (*TCS* triggering message): this produces the unifier $\lambda' = [A_{CFB} = A_{TCS}, C_{CFB} = B_{TCS}]$. The intentional condition $\{\mathit{idle}(C_{CFB}), \mathit{dialing}(A)\}$ is also compatible with the *TCS* activation condition.

The resulting interaction provides us with the call configuration $CC' = \{A : \mathit{POTS}, B : \mathit{CFB}(C), C : \mathit{TCS}(A), C : \mathit{POTS}\}$, with the triggering message $TM' = A.\mathit{call}(B)$ (corresponding to the intentional message $A.\mathit{call}(B)$) and the triggering conditions $TC' = \{\sim \mathit{idle}(B), \mathit{idle}(C), \mathit{dialing}(A)\}$.

The remaining question concerns the visibility of the detected indirect interaction. Clearly, the direct triggering of *CFB* and the indirect one of *TCS* lead to two different expected network answers. It can be established by the comparison of the activation diagrams. Thus, there is a visible interaction with this configuration.

Concerning the second activation branch of *CFB*, the resulting interaction is quite similar. Only the condition changes. The new triggering condition is $\{\sim \mathit{idle}(B), \sim (\mathit{idle}(C)), \mathit{dialing}(A)\}$. The comparison of the central answer and of the activation diagrams allows us to conclude this interaction is visible too: with *CFB*, A would emit a busy line tone and with *TCS* the refusal message. This interaction is visible too.

To summarize, the previous computations simply allow us to retrieve the well known interactions between *TCS* and *CFB* which are detected by the programs giving the following as result:

CC	$\{A : \mathit{POTS}, B : \mathit{CFB}(C),$ $B : \mathit{TCS}(A), C : \mathit{POTS}\}$
TM	$A.\mathit{call}(B)$
TC or	$\{\sim \mathit{idle}(B), \mathit{idle}(C), \mathit{dialing}(A)\})$ $\{\sim \mathit{idle}(B), \sim \mathit{idle}(C), \mathit{dialing}(A)\})$

CC’	$\{A : \mathit{POTS}, B : \mathit{CFB}(C),$ $C : \mathit{TCS}(A), C : \mathit{POTS}\}$
TM’	$A.\mathit{call}(B)$ (int. message : $A.\mathit{call}(C)$)
TC’ or	$\{\sim \mathit{idle}(B), \mathit{idle}(C), \mathit{dialing}(A)\}$ $\{\sim \mathit{idle}(B), \sim \mathit{idle}(C), \mathit{dialing}(A)\}$

All the triggering informations (subscription configuration, state conditions, triggering message, status of visibility) are automatically computed from the given of the service diagrams and their associated constraints. The results given above have been computed by our Prolog prototype.

5 Conclusion

In this paper, we presented a formalism dedicated to service specification and a static interaction detection method implanted for such specifications. The detection method uses two algorithms implemented in Prolog. The first one compares a service specification to the basic system specification in order to find the triggering information of the service, that is to say, the conditions of its activation. The second one compares the triggering informations of two services in order to find interactions between them. These two algorithms have been shortly introduced and performed on a classical example (interactions between CFB and TCS). A comparison with other works on interaction detection can be found in [13] and concerns all services given in the FIW98 and FIW00 contests, except those dealing with billing. In order to improve our detection method, we could perform further works in several directions. First, the calculus of the triggering information allows to compute an activation path (from the initial node to the activation point) in the diagram that could be exploited. For example, giving triple <triggering path, triggering message,triggering condition> can be seen as giving a (uninstantiated) test case. Combined with the use of the constraints, all this could be used in order to determine state reachability in the model. The service specifications could also be extended by adding data types definition (and their associated operations) or by adding some other kind messages and equipments or else by distinguishing phones and users to model services based on a notion of user mobility.

Acknowledgements. We thank Francis Klay for discussions, and for financial support of France Telecom with the MOECIF project. We also thank the French national project RNRT ValiServ.

References

1. R. Accorsi, C. Areces, W. Bouma, and M. De Rijke. Features as Constraints. In *[7]*, pages 210–225, 2000.
2. D. Amyot, L. Charfi, N. Gorse, and T. Gray. Feature Description and Feature Interaction Analysis with Use case Maps and LOTOS. In *[7]*, pages 274–289, 2000.
3. D. Amyot and L. Logrippo, editors. *Feature Interactions in Telecommunications and Software Systems 7*. IOS Press, 2003.
4. K. Berkani, R. Cave, S. Coudert, Francis Klay, P. Le Gall, F. Ouabdesselam, and J.-L. Richier. An environment for interactive service specification. In *[3]*, pages 25–41, 2003.
5. L.G. Bouma and K. Kimbler, editors. *Feature Interactions in Telecommunications and Software Systems 5*. IOS Press, 1998.
6. L. D Bousquet, F. Ouabdesselam, J.-L. Richier, and N. Zuanon. Incremental Feature Validation : A synchronous Point of View. In *[5]*, pages 262–275, 1998.
7. M. Calder and E. Magill, editors. *Feature Interactions in Telecommunications and Software Systems 6*. IOS Press, 2000.
8. D. Cansell and D. Méry. Abstraction and refinement of features. In *[10]*, pages 65–84, 2000.
9. R. Crespo, L. Logrippo, and T. Gray. Feature Execution Trees and Interactions. In Hamid R. Arabnia, editor, *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, PDPTA '02,* pages 1230–1236, 2002.
10. S. Gilmore and M. Ryan, editors. *Language Constructs for Describing Features*. Springer-Verlag London Ltd, 2000.
11. N. Griffeth, R. Blumenthal, J.-C. Gregoire, and T. Ohta. Feature Interaction Detection Contest. In *[5]*, page 327, 1998.
12. M. Heisel and J. Souquières. A Heuristic Algorithm to Detect Feature Interactions in Requirements. In *[10]*, pages 143–162, 2000.
13. H. Jouve. *Caractérisation et détection des interactions à partir de spécification graphiques*. PhD thesis, Université d'Evry, 2003. forthcoming thesis.
14. F. Klay, M. Rusinovitch, and S. Stratulat. Analysing feature interactions with automated deduction systems. In *7th International Conference on Telecommunication System Modeling and Analysis*, 1999.
15. M. Kolberg, E. H. Magill, D. Marples, and S. Reiff. Second Feature Interaction Contest. In *[7]*, pages 213–231, 2000.
16. M. Nakamura, Y. Kakuda, and T. Kikuno. Feature Interaction Detection Using Permutation Symmetry. In *[5]*, pages 187–201, 1998.
17. M. Plath and M.D. Ryan. Defining features for CSP: Reflections on the feature interactions contest. In *[10]*, pages 163–176, 2000.
18. M. Plath and M.D. Ryan. The feature construct for SMV : semantics. In *[7]*, pages 129–144, 2000.
19. S. Reiff. Notes on Call Configurations with Features. In *[10]*, 2000.
20. D. Samborski. Stack Service Model. In *[10]*, 2000.
21. T. Ohta T. Yoneda. A Formal Approach for Definition and Detection of Feature Interaction. In *[5]*, pages 202–216, 1998.
22. K.J. Turner. Formalising the chisel feature notation. In *[7]*, pages 241–256, 2000.
23. T. Yoneda, S. Kawauchi, J. Yoshida, and T. Ohta. Formal approaches for Detecting Feature Interactions, Their Experimental Results, and Application to VoIP. In *[3]*, pages 205–212, 2003.

Service Discovery in Mob$_{adtl}$

C. Montangero, L. Semini, S. Semprini

Dipartimento di Informatica, Università di Pisa

Abstract. Service discovery mechanisms play an increasingly relevant role in the definition and construction of a flexible and powerful infrastructure for applications that have the Internet as their reference running environment. Therefore it is not only sensible but highly desirable and somehow mandatory, to address the unambiguous and rigorous description of service based systems, in the context of the research on formal methods for the specification and verification of distributed systems. Besides, it would be greatly useful to have a common framework on which to base the specification of distributed systems in general and of services based systems in particular. In this paper we claim that Mob$_{adtl}$, with its underlying model, its logic and its refinement-oriented specification process, can be such a framework and show some evidence of how it can be used in this sense.

1 Introduction

In the search for an ever more flexible and powerful net infrastructure to ease the interaction of applications on WANs (and the Internet as the main reference scenario), services play a relevant role. Applications that need some particular information or that, more in general, need to perform tasks for which they are not equipped should be granted the chance to look for other applications able to fulfil their requests. In a classic distributed environment, where everything can be thought of as reasonably stable and stationary, such service providers would be identified once and for all and newbie would be promptly presented to all the inhabitants of a net. In the new settings, where services are no longer stable nor stationary, more powerful means for their discovery are needed.

Besides, in such a setting, being able to unambiguously state the properties exhibited by our systems is of the upmost importance. To achieve the proper level of confidence on the claims about the properties of interest, we need proper formal tools that allow us to specify our systems and to reason on these specifications. The specification of services discovery mechanisms is just an instance of this wider picture.

Service discovery mechanisms are a relevant part in the life of a distributed system, it would be a great benefit to be able to accommodate in the same framework both the description of these mechanisms and the specification of distributed systems so to ease the integration of the two parts. Another appealing feature would be the specification of a general service discovery mechanism, to avoid premature choices with respect to service implementations. This would give the designers the chance to keep different concerns separate, namely the specific system architecture and functionalities on the one side and the service discovery mechanisms on the other. Of course, when the time for a choice has come, it should be possible to detail the general description to match a particular mechanism and still keep the general description as a valid interface between the system and the service providers.

Our claim is that Mob$_{adtl}$ is such a framework, i.e. Mob$_{adtl}$ offers all we need to specify the exploitation of service discovery mechanisms in the most general setting of mobile distributed systems running on WANs, and meets the requirements elicited so far.

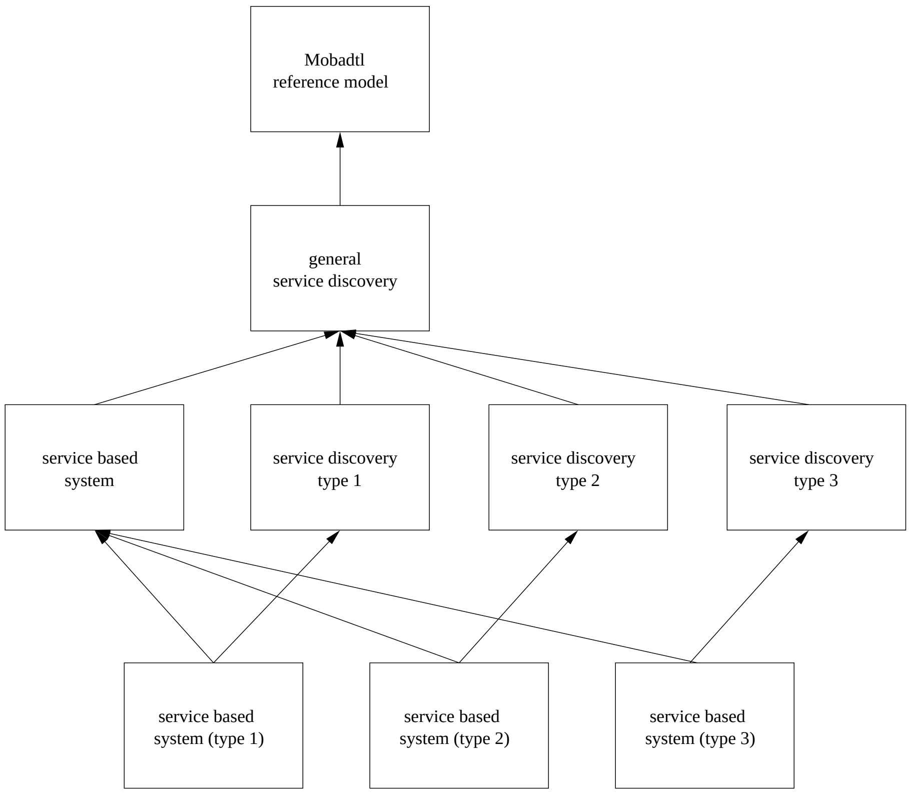
The Mob$_{adtl}$ methodology is based on a model that captures a general structure apt to be used to describe mobile distributed systems. This model poses the bases for the whole methodology by defining a set of concepts: agents (mobile or stationary computational components), neighborhoods (the localities, the places where agents live), and guardians (the entities that embody and enact the security policies relative to each neighborhood and practically take care of communications and mobility handling agents' requests). The Mob$_{adtl}$ model is meant to be a general well defined architecture, where several aspects, e.g. routing for the delivery of messages, are not specified in detail. This allows accommodating any choice that could arise from specific needs of a system. Being this model the very base of the methodology, it is the notion of refinement that guides the designer in the description and verification of a Mob$_{adtl}$ system.

Taking the move from a description of the reference model, passing through several intermediate steps, a specification of the desired system will be produced that gives details for what is intentionally left under–specified in the reference model. These details complete the specification with all the peculiar functional and non–functional aspects of the system of interest. A proper, rigorous definition of a refinement relation guarantees that the refinement process preserves the validity of properties from step to step. Such refinement relation relies on logical implication between specifications written in DSTL(x), a logic that allows describing temporal properties of distributed states of components communicating asynchronously, thus giving the means to talk about the evolution of the computations of distributed systems.

The idea underlying this work is the following: we refine the Mob$_{adtl}$ model to describe a general abstract interface to service discovery mechanisms, thus actually defining a new reference model that is to be considered as the starting point of the specification of any mobile distributed system that is somehow related to services. This new reference model, together with the existing methodology, is the common framework we are looking for. Then, via proper refinement process, we can describe any particular service discovery mechanism keeping as the interface the abstract description we gave at the beginning.

The compositional nature of Mob$_{adtl}$ specifications allows us to think of a scenario like the one represented in Figure 1. Given the specification of a service based system, defined in terms of the reference model, we can derive an instanti-

Fig. 1. A scenario of refinement and composition.



ation of the system depending on the particular discovery mechanism we choose to pick up from a library of mechanism specifications.

In Figure 1, boxes represent specifications and arrows represent refinement relations between specifications.

In this paper, for the sake of economy in the presentation of the logic and of the model, we focus our attention on the specification of service discovery mechanisms rather than on mobile systems. We report a particular refinement of the general Mob$_{adtl}$ model that constrains it to a stationary setting where security concerns, of the upmost relevance to the general Mob$_{adtl}$ model, are not taken into account. Porting the results presented in this paper to the general model would be an easy task. This porting would imply the exploitation of DSTL(x) full expressiveness, that is needed for a proper description of the mechanics of the agent/guardian relationship, and of communications and mobility.

2 Specification and refinement in DSTL(x)

We write our specifications using an asynchronous, distributed, and temporal logic called DSTL(x), the first order extension of DSTL [16]. It is based on a logic for distributed states, DSL [17], and includes temporal operators à la Unity [7]. We build on the Unity logic since it is only a subset of linear time temporal logic, with a limited number of operators, which are rather easy to understand intuitively, and cannot be nested. However, the logic permits to express most of the important properties of a system, exploiting the simplicity and proof straightforwardness which makes Unity more appealing for the designers than linear time temporal logic. The extensions we make permit to deal with distributed asynchronous systems. We name the components of the system and relate properties which might hold in distinguished components, in an asynchronous setting.

We assume a denumerable set of component names $\{m, n, m_1, m_2, \ldots\}$, and that the variables set includes a denumerable set of component variables $\{M$, N, M_1, $M_2, \ldots\}$.

We introduce location modalities for each component in a system: we use component names, with a different font. For instance, $\mathsf{m_1}$ is the location modality corresponding to component m_1, and $\mathsf{m_1} Iam(m_1)$ stays for "in component m_1, $Iam(m_1)$ holds". We let quantifiers range over modalities, and M , N , $\mathsf{M_i}$ ... are location modality variables. Binding between location variables and regular variables is possible. For example, $\forall M.\, \mathsf{M} Iam(M)$ means that for all components m_i, $\mathsf{m_i} Iam(m_i)$ holds. Quantification over modality variables is done in a standard way, following, for instance [9].

Due to focus and to space reasons, in this work we consider a fragment of DSTL(x), and do not dwell in the formal aspects of the logic. The reference paper is [16], where relationship to many logic for distributed and mobile systems is also discussed. Here we just recall that the main novelty of the logic is an innovative semantic domain: the Kripke models are built on worlds that are arbitrary sets of computation states, instead of single states or tuples of them (one for each component). These choices, adopted in many logics for distributed systems are not well suited to asynchronous communications.

2.1 Syntax

$$F \;::=\; A \mid \bot \mid\; \sim F \mid F \wedge F' \mid \mathsf{M_i} F$$
$$\phi \;::=\; \forall\exists F \mid F \textsc{ leads\_to } F' \mid F \textsc{ because } F' \mid \textsc{stable } F \mid \textsc{init } F$$

The first equation defines DSL(x) formulae: A is an atom, $\bot$ is the propositional constant *false*. With $\bar{\mathsf{M}}_\mathsf{i}$ we denote the dual of $\mathsf{M_i}$, i.e., $\bar{\mathsf{M}}_\mathsf{i} F \equiv \sim \mathsf{M_i} \sim F$. With $\top$ we denote *true*, i.e. $\top \equiv \sim \bot$.

The second equation defines DSTL(x) formulae: A DSL(x) formula is also a DSTL(x) formula, provided it is a closed sentence, in prenex normal form. For the sake of readability, we leave universal quantification implicit, and make explicit, when needed, existential quantifiers. For instance, $\exists y.\, \mathsf{M} p(x) \rightarrow q(x, y)$, is implicitly prefixed by $\forall M, x$. Conversely, quantification of temporal formulae is always implicit, as discussed below.

Operator LEADS_TO expresses a liveness condition, and is similar to Unity's $\mapsto$ (leads to): F is always followed by F'; BECAUSE expresses a safety condition, and says that F must be preceded by F'; STABLE extends Unity's STABLE to the distributed case; INIT permits to describe the initial state.

Temporal formulae are written without explicit quantification. The intended meaning is that a formula F is universally quantified over all values of the variables appearing in the premises of F, and existentially quantified on the remaining variables. For example $\mathsf{M}\, p(x) \wedge q(y)$ LEADS_TO $\mathsf{N}\, r(x, M, z)$ should be understood as prefixed by $\forall\, M, x, y\ \exists\, N, z$. The variables in the formulae with INIT and STABLE are all universally quantified.

The domain over which a variable is quantified (i.e. its sort) can be understood from the context or explicitly defined. What is important is that we assume these domains to be invariant during time and in space.

2.2 Semantics

The models for DSTL(x) formulae are built on structures like the one in the following figure, which describes the computation of a system with two components. Here, p, q, ... are the properties holding in the states.

$$\begin{tikzcd}
(m) & p \arrow[r] & q \arrow[rr] \arrow[dr] & & r \arrow[rr] & & u,z \arrow[r] & z \arrow[r] & z \arrow[r] & {} \\
(n) & p,t \arrow[r] & u \arrow[r] & v \arrow[r] & p \arrow[r] & w,t \arrow[ur] \arrow[rrr] & & & w,t \arrow[r] & {}
\end{tikzcd}$$

We call S_i the set of states of component m_i, and S the set of all the states of the computation. A distributed state ds is any subset of S, and ds^0 is the set of the initial states.

Let $\mathcal{M}$ be a model, the semantics of DSTL(x) formulae is given by structural induction in Table 1.

Note that a distributed state is any set of states. This means that when we have to check a condition like $\forall ds \ldots \exists ds' \ldots$, we need to consider all possible subsets of S. This may lead to counter–intuitive choices, like taking larger states than needed. However, the specifier can be safely guided by the natural interpretation of the operators. Anyhow, our definition of distributed state is exactly what was needed to overcome the problems with the existing logics for distributed systems, which do not have the right expressive power to reason on the systems behaviour, when the communication is based on asynchronous message passing.

Example 1. We discuss the satisfiability of some formulae with respect to the computation above.

Table 1. Semantics of DSTL(x).

$\mathcal{M} \models_T F$	iff	$\forall ds.\, ds \models F$
$\mathcal{M} \models_T F \ \textsc{leads\_to}\ F'$	iff	$\forall ds.\, ds \models F$ implies $\exists\, ds' \geq ds.\, ds' \models F'$
$\mathcal{M} \models_T F \ \textsc{because}\ F'$	iff	$\forall ds.\, ds \models F$ implies $\exists\, ds' \leq ds.\, ds' \models F'$
$\mathcal{M} \models_T \ \textsc{stable}\ F$	iff	$\forall ds.\, ds \models F$ implies $\exists ds' >_c ds.\, ds' \models F$
$\mathcal{M} \models_T \ \textsc{init}\ F$	iff	$ds^0 \models F$

where:

$ds \models \top$		
$ds \models A$	iff	A holds in s for all $s \in ds$
$ds \models \sim F$	iff	not $ds \models F$
$ds \models F \wedge F'$	iff	$ds \models F$ and $ds \models F'$
$ds \models \mathsf{m}_i F$	iff	$\exists s \in ds \cap S_i$ and $\{s\} \models F$

We say that $ds \geq ds'$ ($\leq$) if and only if each state in ds' is followed (preceded) by a state in ds and each state in ds is preceded (followed) by a state in ds'. Relations $\geq$ and $\leq$ are reflexively and transitively closed. We say that $ds >_c ds'$ when the states in ds' are immediately followed by a state in ds (and viceversa) and ds does not include ds'.
We recall that a DSTL(x) formula is a closed sentence, shaping (implicitly or explicitely) $\forall \ldots \exists \ldots \phi$. Hence, we restrict the semantic definitions to the (ground) subformula ϕ.

$\mathcal{M} \models_T \mathsf{n}u \ \textsc{leads\_to}\ \mathsf{m}u \wedge \mathsf{n}t$**.** A distributed state satisfies the premise if it contains, for instance, the second state of component n, where u holds, call it s. It is immediate to find a distributed state following $\{s\}$ and satisfying the consequence, e.g. the first pair of states where u holds in m and t holds in n (related by a communication in the figure), call this pair p. Any larger distributed state ds including s satisfies the premise and is followed, for instance by $ds \cup p$, which satisfies $\mathsf{m}u \wedge \mathsf{n}t$.

$\mathcal{M} \models \mathsf{n}w \ \textsc{because}\ \mathsf{n}p \wedge \mathsf{n}u$**.** Consider, for instance, the first state, say s, of n where w holds. Set $\{s\}$ is preceded by the pair p composed of the initial and the second state of n. Set p satisfies the consequence. A superset ds of $\{s\}$ is preceded by $ds \cup p$.

$\mathcal{M} \not\models \mathsf{n}w \ \textsc{because}\ \mathsf{n}(p \wedge u)$**.** To satisfy the formula, we would need a singleton state of n satisfying both p and u.

$\mathcal{M} \models \ \textsc{stable}\ \mathsf{n}(w \wedge t)$**.** Any distributed state including a state of n satisfying $w \wedge t$, is immediately followed by a distributed state including a state of n satisfying $w \wedge t$.

$\mathcal{M} \models \exists M.\, \bar{\mathsf{M}}(u \to z)$**.** There exists a component, m in our example, where each state satisfies: $u \to z$.

2.3 Refinement and the proof assistant Mark

In the next section we describe the pragmatic issues of the Mob$_{adtl}$ refinement process. From a foundational point of view, a theory $\mathcal{T}$ (a set of DSTL(x) formulae) is refined by theory $\mathcal{T}'$ if all the formulae in $\mathcal{T}$ can be derived from the formulae in $\mathcal{T}'$, according to the axiom system of the logic. Examplar axioms and rules of DSTL(x) is in Tables 5 and 6 in the appendix.

Refinement verification is supported by the proof assistant Mark (Mob$_{adtl}$ Reasoning Kit) [12], which is developed in Isabelle [19]. The advantages of using Mark include: the ability to keep track of the assumptions a property relies on; the ability to handle standard bureaucratic activities (boolean reasoning and standard verification techniques); the ability to support the emergent proving patterns; the ability of introducing tactics, i.e. proof patterns, a powerful mechanism to manage the complexity of proofs.

3 Mob$_{adtl}$

In this section we give a brief description of the model and an extract from its axiomatization, to give an idea of what Mob$_{adtl}$ specifications look like and how things work. Moreover we clarify the refinement process that is mentioned in the Introduction and that is the leit–motif of Mob$_{adtl}$ methodology in general, and the rational behind the work presented in this paper in particular.

Mob$_{adtl}$ specifications describe the behaviour of agents, i.e. running computations, that roam in a network of bounded localities, called neighborhoods, and communicate asynchronously. A stationary entity called guardian is attached to each neighborhood. It is in charge of monitoring the activities of the agents that interact with the neighborhood, either living in the neighborhood or trying to communicate with agents inside it. Each agent willing to perform some action, e.g. communicating or moving, must refer to the guardian of the neighborhood where it is currently located. Each guardian manages the requests of its agents basing its decisions on the security policy of the neighborhood it is guarding. It may happen that the interaction of several guardians is required in order to fulfill an agent request. The result of this process is a global dynamic security policy based on the composition of several local security policies.

Figure 2 gives a pictorial representation of these concepts. The figure shows a neighborhood like a container of agents controlled by a guardian which communicate with other guardians to deal with a communication request.

The refinement process allows the designer to describe the properties of interest at different level of details, from a very general, abstract specification down to a specification of the features of the system, as detailed as needed.

At each level of the refinement chain, the specification of a system is structured as a triple of theories specifying the behaviour of agents, the behaviour of guardians and the interactions among agents and guardians. Theories are structured so that the specification of each class of homogeneous components is kept separate from the specification of other classes, and can be refined independently, in a fully compositional way. The specification of a class contains only

Fig. 2. The Mob$_{adtl}$ model.

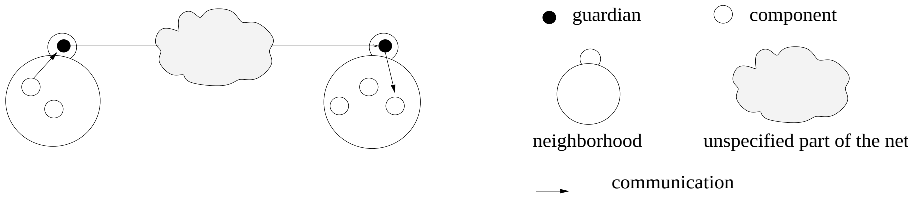
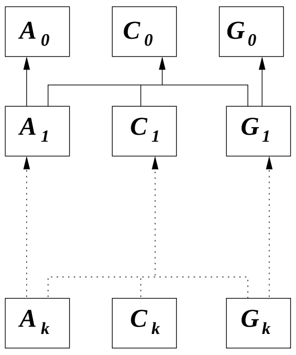


Fig. 3. Design methodology: a theory hierarchy.



local axioms or axioms relating events and conditions in different components of that class. The coordination axioms specify interactions involving components from different classes. At each refinement step it is possible to extend the set of axioms to specify new properties, or to refine and distribute the formulae in the existing theories. The final goal of the refinements is to specify the classes fully and to leave in the coordination theory only the assumptions on the underlying middleware or the properties that will have to be guaranteed by the application context. A typical middleware property is that point–to–point communication is reliable; assumptions on the context can include human behaviour. Figure 3 shows the overall refinement/extension structure of the Mob$_{adtl}$ theories.

At level zero we define the global properties of the model: agents and guardians are the classes of interest, $\mathcal{A}_0$, $\mathcal{G}_0$ their specifications, and $\mathcal{C}_0$ specifies the coordination among them. We define the refinement relation between theories accordingly to the proposed refinement strategy. We ask that at each step theory $\mathcal{A}_i$ refines $\mathcal{A}_{i-1}$, $\mathcal{G}_i$ refines $\mathcal{G}_{i-1}$, and $\mathcal{A}_i \cup \mathcal{C}_i \cup \mathcal{G}_i$ refine $\mathcal{C}_{i-1}$.

Since the focus of this paper is not on the Mob$_{adtl}$ model but on services, Table 2 presents only the axioms of level 0 dealing with communications. Moreover, since the purpose is just to give the taste of Mob$_{adtl}$ specifications and not to burden the reader with technicalities, a simplified version is presented that does not take into account security and failures. The complete specification can be found in [13].

In this table and in the following ones, location variable G ranges over the guardian names, and S , R and C range over the agent names.

Table 2. Mob$_{adtl}$ communications.

$\mathcal{A}_0$		
$\mathcal{C}_0$	**C:**	S out(M,P) LEADS_TO R in(M,S)
	C′:	R in(M,S) BECAUSE G (msgReq(M,S,P) $\wedge$ satisfy({R},P))
	Id$_2$:	G msgReq(M,S,P) BECAUSE S out(M,P)
$\mathcal{G}_0$	**Sat$_1$:**	$\bar{\mathsf{G}}$ satisfy(Set, P) $\wedge$ satisfy(Set′, P) $\leftrightarrow$ satisfy(Set $\cup$ Set′, P)
	Sat$'_1$:	$\bar{\mathsf{G}}$ satisfies(S, P) $\leftrightarrow$ satisfy({S}, P)
	Sat$_2$:	$\bar{\mathsf{G}}$ satisfies(S, P) $\wedge$ satisfies(S, P′) $\leftrightarrow$ satisfies(S, P$\cup$ P′)
	Sat$'_2$:	$\bar{\mathsf{G}}$ satisfies_constraint(S, Cons) $\leftrightarrow$ satisfies(S, {Cons})

In the state of an agent, predicate out(M,P) means that message M has been sent to a receiver that satisfies profile P, predicate in(M,S) represents a message M received from agent S, and axiom **C** says that any invoice will result in a delivery (in this simplified model). Axioms **C′** guarantees that a message is received only if actually sent, and that the receiver satisfies the profile. Finally, axioms **Id2** says that there can be a communication request attributed to a sender S only if S actually committed to communicate: in a guardian msgReq(M,S,P) specifies the message body, the sender and the profile of the receiver.

Profiles are the solution that Mob$_{adtl}$ gives to the need of a general, multi–purpose mechanism to identify sets of entities satisfying given properties, e.g. QoS constraints, identity constraints, security constraints etc. In the general Mob$_{adtl}$ model, profiles are used to identify the set of the possible receivers of a communication (cfr. axiom **C′**) and the possible targets of a movement.

Predicate satisfy relates sets of names and profiles. A profile is specified as a set of constraints, and distinct profiles can have overlapping constraints. Predicate satisfy(Set, P) says that the elements of Set satisfy profile P. These sets are built exploiting the relations defined by predicates satisfies and satisfies_constraint following this pattern: axioms **Sat$'_2$** says that a single entity can satisfy a single profile and introduces satisfaction of a set of profiles, as stated in axiom **Sat$_2$**; set of entities are dealt with similarly in axioms **Sat$'_1$** and **Sat$_1$**.

In this paper, profile satisfiability plays a central role as the very mechanism at the basis of service discovery. A service is defined by a set of functional and non functional parameters built up in a profile. A service request is successfully resolved when an entity is found that provides the service specified in the profile contained in the request.

4 The service discovery interface

In this section we describe the process that leads us from the specification of Mob$_{adtl}$ to the Mob$_{adtl}$ service discovery interface (Mob$_{adtl}$ SDI), i.e. the second box in the refinement hierarchy shown in Figure 1. The axioms of each refinement step are given and properly commented, and the satisfaction of the refinement relation between different steps is verified and explained with references to the rules of DSTL(x).

4.1 Requirements for service discovery

At the first level, referred to as Level 1 and shown in Table 3, we simply state the overall properties that we think sufficient and necessary to specify a service discovery mechanism interface. Following the Mob$_{adtl}$ methodology, we do not copy the axioms from the previous level, that are left untouched and not refined. In this case, all the axioms in Table 2 are implicitly imported in this level.

Table 3. Requirements of the Mob$_{adtl}$ SDI.

$\mathcal{A}_1$	**Ser**$_{1.1}$: $\bar{\mathsf{C}} \sim$ (service_available(S, Fun, Pars) $\wedge$ no_service_available(Fun, Pars))
	Ser$_{1.2}$: C out(ser(Fun,Pars),{name(S)}) BECAUSE C service_available(S,Fun,Pars)
$\mathcal{C}_1$	**Ser**$_{1.3}$: C service_req(Fun,Pars) LEADS_TO C (service_available(S, Fun, Pars) $\vee$ no_service_available(Fun, Pars))
	Ser$'_{1.3}$: C (service_available(S, Fun, Pars) $\vee$ no_service_available(Fun, Pars)) BECAUSE C service_req(Fun,Pars)
	Ser$''_{1.3}$: C service_available(S, Fun, Pars) BECAUSE G satisfy({S},{service_profile(Fun,Pars,C)})
	Ser$'''_{1.3}$: C no_service_available(Fun, Pars) BECAUSE G unsatisfiable({service_profile(Fun,Pars,C)})
$\mathcal{G}_1$	**Sat**$_3$: $\bar{\mathsf{G}}$ unsatisfiable(P) $\rightarrow \sim$ satisfy(Set, P)

First of all, it is natural to require agents to have consistent knowledge about service providers. This is stated in **Ser**$_{1.1}$: service_available(S,Fun,Pars) means that the discovery service has notified component C that S can provide service Fun satisfying the constraints in Pars; viceversa, no_service_available(Fun,Pars) means that the search for the specified service failed. A more complete specification would have one more parameter, to carry information about the reasons of the failure.

Axiom **Ser**$_{1.2}$ says that in order to ask S for a particular service, a component must know that S provides that service. These are properties dealing with the local state of components. All the other axioms of this level belong to the coordination theory, since they are just the most abstract specification of a process that leads to the discovery of services and that can involve both guardians and components.

$\mathbf{Ser}_{1.3}$ models the outcome of a service request: either the requesting component will have the identity of a service provider, or it will be informed that there is no suitable provider. Of course, such a knowledge can be acquired only if the proper request has been expressed, as stated in axiom $\mathbf{Ser}'_{1.3}$. The last two axioms $\mathbf{Ser}''_{1.3}$ and $\mathbf{Ser}'''_{1.3}$ express the discovery strategy, i.e., that it is the responsibility of the guardians to find a provider that satisfies the profile in the request. The name of the requesting component is put in the profile, since security or authentication concerns may determine the response to service requests.

$\mathbf{Sat}_3$ is an auxiliary axiom to avoid to use ∼satisfy in a context where variable Set would be (erroneously) existentially quantified, like in $\mathbf{Ser}'''_{1.3}$.

4.2 Refinement of the service discovery model

The first refinement takes us to Level 2, a definition of the role of guardians in the process of service discovery. The related axioms are reported in Table 4. As previously stated, this is the level to start specifying specific discovery mechanisms and service based distributed systems.

The idea is that when a component needs to look for a service, a guardian will be informed (axioms $\mathbf{Ser}_{2.1}$) and will take the responsibility of identifying the proper service provider, if any (axiom $\mathbf{Ser}_{2.5}$). Of course a guardian will look for a service provider only if a component asks for it (axiom $\mathbf{Ser}'_{2.1}$), and a component will receive the result of a service request only if a guardian has received that request (axiom $\mathbf{Ser}_{2.4}$).

Table 4. Refinement of the Mob$_{adtl}$ SDI.

$\mathcal{A}_2$	
$\mathcal{C}_2$	$\mathbf{Ser}_{2.1}$: C service_req(Fun,Pars) LEADS_TO G service_resolve(Fun,Pars,C)
	$\mathbf{Ser}'_{2.1}$: G service_resolve(Fun,Pars,C) BECAUSE C service_req(Fun,Pars)
	$\mathbf{Ser}_{2.2}$: G service_found(Fun,Pars,C,S) LEADS_TO C service_available(Fun,Pars,S)
	$\mathbf{Ser}_{2.3}$: G no_service_found(Fun,Pars,C) LEADS_TO C no_service_available(Fun,Pars)
	$\mathbf{Ser}_{2.4}$: C service_available(Fun,Pars,S) ∨ no_service_available(Fun,Pars) BECAUSE G service_resolve(Fun,Pars,C)
$\mathcal{G}_2$	$\mathbf{Ser}_{2.5}$: G service_resolve(Fun,Pars,C) LEADS_TO G service_found(Fun,Pars,C,S) ∨ no_service_found(Fun,Pars,C)
	$\mathbf{Ser}'_{2.5}$: G service_found(Fun,Pars,C,S) ∨ no_service_found(Fun,Pars,C) BECAUSE G service_resolve(Fun,Pars,C)
	$\mathbf{Ser}_{2.6}$: $\bar{\text{G}}$ service_found(Fun,Pars,C,S) → satisfy({S},{service_profile(Fun,Pars,C)})
	$\mathbf{Ser}_{2.7}$: $\bar{\text{G}}$ no_service_found(Fun,Pars,C)→unsatisfiable({service_profile(Fun,Pars,C)})

Axioms $\mathbf{Ser}_{2.6}$ and $\mathbf{Ser}_{2.7}$ describe how the general service discovery mechanism is based on the profile resolution capabilities of guardians already introduced in the general Mob$_{adtl}$ reference model.

The axioms at this level include those in Table 4 and those in the previous Tables, but $\mathbf{Ser}_{1.3}$ and $\mathbf{Ser}'_{1.3}$. $\mathbf{Ser}_{2.1}$ and $\mathbf{Ser}_{2.5}$ refine $\mathbf{Ser}_{1.3}$, by transitivity (rule LTR). Similarly, $\mathbf{Ser}'_{1.3}$ is refined by $\mathbf{Ser}'_{2.1}$ and $\mathbf{Ser}_{2.4}$, via rule BTR.

Exploiting rules LTR, LPD and LWC, we can derive an interesting property holding at this level, namely:

G service_resolve(Fun,Pars,C) LEADS_TO
C service_available(Fun,Pars,S) $\vee$ no_service_available(Fun,Pars)

Notice that at this level of refinement, it is not yet determined how many answers a discovery request will receive, since the number of guardians that are contacted is also not determined, and left to further refinements. Also, a possible refinement is one where the negative answer does not necessarily means that there are no suitable services in the system, but simply that the involved guardians decided not to wait any longer, by some sort of distributed consensus. An example of such a behaviour has been specified in [16].

5 Related Work

SOAP and WSDL are the current core technologies for Web Services [22]. Coupled with a transport protocol, they allow simple point–to–point interactions among distributed software components. However, when complex interactions including many components are needed in a transaction, SOAP and WSDL show their weakness: orchestration, i.e., the combination of composition and coordination, is needed [1, 10].

Mob$_{adtl}$ belongs to the family of orchestration models, it enables the construction of complex applications as the composition of distributed sub services available on the net, possibly without a central coordinator.

To dynamically discover the services that are available over the net, and to combine them on-the-fly, we need to face a number of challenges related to dynamic type checking, service profile publication/retrieval, new binding schemes that cater for QoS characteristics. Some novel approaches that may be formalized as refinement of the Mob$_{adtl}$ SDI, include JXTA [5] and IBM MatchMaking Engine [14] for discovery, and [11, 10, 2, 6, 4] for service composition.

When discovering and composing services available on the net, it is essential to consider security, not to be fooled and connect to malicious servers. Some security policies that could be formalized in Mob$_{adtl}$ SDI are: DNSSEC which uses digital signatures to solve several DNS problems; LDAP over SSL (LDAPS) and HTTP over SSL (HTTPS) which allow secure and authenticated client access to servers; XKMS from OASIS and W3C providing developers with a way to use XML-based transactions to exchange public keys.

The Mob$_{adtl}$ profile mechanism could be used to specify the constraints on the required services, with respect to context awareness and adaptation. For instance, [8, 18, 21] envisage self-adaptive software architectures, platforms such as J2ME [23] support the development of services and components on top of low cost devices with limited computational capabilities, and frameworks such as Ubiquitous Web Application [20] allow services to be adapted to device characteristics.

Adaptation strategies span many options, like exploiting mobile code to modify the number, type or location of components [15], or tailoring the interaction protocols to new contexts, e.g. by modifying the accuracy of the transmitted data, in case of changes in the communication bandwidth, or by introducing security mechanisms, to adapt to insecure environments [3].

6 Conclusions

In this paper we showed how Mob$_{adtl}$, a high level declarative model for the design of mobile and distributed systems, offers full support to specify and reason about service discovery mechanisms.

Mob$_{adtl}$ provides a formal methodology to drive the design of applications based on the notion of structured refinements, and we exploited this methodology to define an abstract interface to service discovery mechanisms, thus actually defining a new reference model that is to be considered as the starting point of the specification of any mobile distributed system that is somehow related to services.

References

1. L.F. Andrade, J.L. Fiadeiro, J. Gouveia, G. Koutsoukos, and M. Wermelinger. Coordination for orchestration. In F. Arbab and C.L. Talcott, editors, *5th Int. Conference on Coordination Models and Languages (COORDINATION 2002)*, volume 2315 of *Lecture Notes in Computer Science*, pages 5–13. Springer-Verlag, 2002.
2. A. Arkin et al. Web Service Choreography Interface 1.0. wwws.sun.com/software/xml/developers/wsci/wsci-spec-10.pdf, July 2002.
3. B. Badrinath et al. A conceptual framework for network and client adaptation. *Mobile Networks and Applications*, 5(4):221–232, Dec. 2000.
4. B. Benatallah and M. Dumas. The self-serv environment for web services composition. *IEEE Internet Computing*, Jan.–Feb. 2003.
5. D. Brookshier, N. Krishnan, D. Govoni, and J.C. Soto. *JXTA: JavaTM P2P Programming.* SAMS Publishing, June 2002.
6. F. Casati et al. Adaptive and Dynamic Service Composition in eFlow. Technical report, Hewlett–Packard Company, Mar. 2000. www.hpl.hp.com/techreports/2000/HPL-2000-39.pdf.
7. K.M. Chandy and J. Misra. *Parallel Program Design: A Foundation.* Addison-Wesley, Reading Mass., 1988.
8. S.-W. Cheng et al. Using architectural style as a basis for self–repair. In *Proc. 3rd Working IEEE/IFIP Conf. on Software Architecture (WICSA 2002)*, pages 45–59, Aug. 2002.

9. T. Costello and A. Patterson. Quantifiers and operations on modalities and contexts. In A.G. Cohn, L. Schubert, and S.C. Shapiro, editors, *KR'98: Principles of Knowledge Representation and Reasoning*, pages 270–281. Morgan Kaufmann, San Francisco, 1998.
10. F. Curbera et al. Business process execution language for web services, version 1.0. www-106.ibm.com/developerworks/webservices/library/ws-bpel/, July 2002.
11. The DAML Services Coalition. Daml–s: Web service description for the semantic web. In *Proc. 1st Int. Semantic Web Conf. (ISWC)*, June 2002.
12. G. Ferrari, C. Montangero, L. Semini, and S. Semprini. Mark, a reasoning kit for mobility. *Automated Software Engineering*, 9(2):137–150, Apr 2002.
13. G. Ferrari, C. Montangero, L. Semini, and S. Semprini. The mob_{adtl} model and method to design network aware applications. Technical Report TR-03-08, Dip. di Informatica, Università di Pisa, 2003. At www.di.unipi.it/ricerca/TR/tr.html.
14. S. Field. A personalised needs oriented approach to insurance product sales. Tech. rep. IBM, 2002. www.zurich.ibm.com/pdf/wme/WME_Insurance_Products.pdf.
15. A. Fuggetta, G.P. Picco, and G. Vigna. Understanding Code Mobility. *IEEE Transactions on Software Engineering*, 24(5):342–361, 1998.
16. C. Montangero and L. Semini. Distributed states temporal logic. CORR Archive: cs.LO/0304046.
17. C. Montangero and L. Semini. Distributed states logic. In *9th International Symposium on Temporal Representation and Reasoning (TIME'02)*, Manchester, UK, July 2002. IEEE CS Press.
18. P. Oreyzi et al. An architecture–based approach to self–adaptive software. *IEEE Intelligent Systems*, pages 54–62, May/June 1999.
19. L. Paulson and T. Nipkow. Isabelle. www.cl.cam.ac.uk/Research/HVG/ Isabelle/.
20. The UWA (Ubiquitous Web Application) Project. http://www.uwaproject.org/.
21. M. Satyanarayanan. Mobile information access. *IEEE Personal Communications*, 3(1), Feb. 1999.
22. M. Stal. Web services: Beyond component-based computing. *Communications of the ACM*, 45(10):71–76, Oct 2002.
23. Sun. Java 2 platform Micro Edition (J2ME). http://java.sun.com/j2me/.

Table 5. Axioms and rules of DSL(x).

FOL	axioms of the 1st order logic	**K**	$\bar{\mathsf{M}}(F \to F') \to (\bar{\mathsf{M}}F \to \bar{\mathsf{M}}F')$
DSL1	$\bar{\mathsf{M}}(\bar{\mathsf{M}}F \leftrightarrow F)$	**DSL2**	$\mathsf{M} \neq \mathsf{N} \to \bar{\mathsf{M}}\bar{\mathsf{N}}\bot$
MP	$\dfrac{F \quad F \to F'}{F'}$	**Nec**	$\dfrac{F}{\bar{\mathsf{M}}F}$

Table 6. Axioms and Rules of DSTL(x).

- **Necessitation.** (We use $\vdash_{DSL}$ and $\vdash_{DSTL}$ for the sake of comprehension).

$$\dfrac{\text{for all } x_1 \ldots x_n \text{ exists } y_1 \ldots y_n \;\; \vdash_{DSL} F}{\vdash_{DSTL} \;\; \exists y_1 \ldots y_n F}\,\textbf{Nec}$$

- **Introduction and Elimination Rules.**

$$\textbf{LI}\;\; F \;\textsc{leads\_to}\; F \qquad \textbf{BI}\;\; F \;\textsc{because}\; F \qquad \dfrac{F}{\textsc{stable}\; F}\,\textbf{SI}$$

$$\dfrac{F \;\textsc{leads\_to}\; \bot}{\sim F}\,\textbf{LE} \qquad \dfrac{F \;\textsc{because}\; \bot}{\sim F}\,\textbf{BE} \qquad \dfrac{\textsc{init}\; \mathsf{M}F \quad \textsc{stable}\; \mathsf{M}F}{\bar{\mathsf{M}}F}\,\textbf{SE}$$

- **Transitivity Rules.**

$$\dfrac{F \;\textsc{leads\_to}\; F' \quad F' \;\textsc{leads\_to}\; G}{F \;\textsc{leads\_to}\; G}\,\textbf{LTR} \qquad \dfrac{F \;\textsc{because}\; F' \quad F' \;\textsc{because}\; G}{F \;\textsc{because}\; G}\,\textbf{BTR}$$

- **Premises and consequences strengthening and weakening.**

$$\dfrac{\exists_{\mathrm{F}}\, G \to F \quad F \;\textsc{leads\_to}\; F' \quad \exists_{\mathrm{G'}}\, F' \to G'}{G \;\textsc{leads\_to}\; G'}\,\textbf{LSW}$$

$$\dfrac{F \;\textsc{leads\_to}\; G \quad F' \;\textsc{leads\_to}\; G}{F \vee F' \;\textsc{leads\_to}\; G}\,\textbf{LPD} \qquad \dfrac{G \;\textsc{leads\_to}\; F \quad G \;\textsc{leads\_to}\; F'}{G \;\textsc{leads\_to}\; F \wedge F'}\,\textbf{LCC}$$

Equivalent rules hold for BECAUSE. In the case of INIT:

$$\dfrac{\textsc{init}\; F \quad F \to G}{\textsc{init}\; G}\,\textbf{IW} \qquad \dfrac{\textsc{init}\; F \quad \textsc{init}\; F'}{\textsc{init}\; F \wedge F'}\,\textbf{IC}$$

- **Notification and Confluence.**

$$\dfrac{F \;\textsc{because}\; G \;\; G \;\textsc{leads\_to}\; \mathsf{M}G' \;\; \textsc{stable}\; \mathsf{M}G'}{F \wedge \mathsf{M}\top \;\textsc{leads\_to}\; \mathsf{M}G'}\,\textbf{Nf} \qquad \dfrac{\textsc{stable}\; \mathsf{M}F \;\; \textsc{stable}\; \mathsf{M}F'}{\mathsf{M}F \wedge \mathsf{M}F' \to \mathsf{M}(F \wedge F')}\,\textbf{Cf}$$

- **Properties of the initial state.**

$$\textbf{I1:}\;\; \textsc{init}\; \mathsf{m}\top \qquad \dfrac{\textsc{init}\; \mathsf{m}F}{\textsc{init}\; \bar{\mathsf{m}}F}\,\textbf{I2} \qquad \dfrac{\textsc{init}\; \bar{\mathsf{m}}F}{\textsc{init}\; \mathsf{m}F}\,\textbf{I3}$$

We recall that non temporal DSTL formulae are implicitly universally quantified in all variables with the exception of those explicitely bound by existential quantification. Here, in the case of implication, with $\exists_{\mathrm{G}}\, F \to G$ we denote a formula universally quantified in all the variables of premise F, and existentially quantified in all the free variables of the consequence G.