TINA Service Validation: The ErnesTINA Project*

X. Logean *F. Dietrich* *J.-P. Hubaux*

Institute For Computer Communications and Applications
Swiss Federal Institute of Technology
Lausanne, Switzerland
http://icawww.epfl.ch

Abstract

While extensive work has been carried out with the goal of validating the TINA architecture and the TINA documents, little has been done yet for the validation of TINA services. This is the main focus of the ErnesTINA project. In the ErnesTINA project, we propose an integrated approach to facilitate the validation of TINA services by verifying at run-time that the service implementation has not violated and is not violating certain predefined properties. In this paper, we present the specification of the properties, the run-time observation of the distributed environment, the validation of the properties and finally the implementation of the concepts in a prototype.

1 Introduction

The design and implementation of telecommunication services is a complex task. This task is filled with pitfalls causing errors which may only manifest themselves when the final implementation is observed at run-time. Often, there are many less obvious symptoms which pass unheeded by the human overseers whose job it is to constantly watch for imminent disaster. Entire telecommunication systems can be brought to their knees by an unintended feature interaction, or a misimplemented piece of the greater puzzle. This puzzle, and thus the risk of errors, is likely to get more complicated in the future.
The Telecommunications Information Networking Architecture (TINA) is being proposed by a world-wide consortium to manage the complexity of designing and implementing telecommunications systems safely.

Despite the advantages that can be realized by using the TINA architecture, there have been few answers to the question: how can we be sure that a given TINA compliant implementation behaves as expected?

In the ErnesTINA project we are developing a way to (semi)automatically generate an implementation that observes the dynamic behavior of a telecommunication system, maintaining a notion of whether or not that behavior violates some predefined properties. Therefore, we are concentrating on the twofold problem of specification and validation of telecommunication services under the TINA architecture; what behavior needs to be observed at run-time, how is that behavior specified and how is the automatization based on that specification to be achieved?
The remainder of this paper is structured as follows. In Section 2, we explain the approach taken in the ErnesTINA project. In Section 3, we describe the properties we are considering for validation. Section 4 illustrates how those properties can be validated at run-time which is followed by a case study in Section 5. Finally, our conclusions and an outlook on ongoing work is presented.

2 General Description

In our approach we express properties that the service should satisfy (i.e., desired behavior) or not violate (i.e., unwanted behavior). Those properties are expressed in an implementation language independent manner but validated at the implementation level when the service is running. The link between the implementation language independent character of the properties and the validation of those properties at run-time is, in our work, provided by the Object Definition Language (ODL) [9].
The TINA Consortium proposes ODL for the specification of computational objects. An ODL specification provides an abstract view about the actual service implementation by specifying the object- and interface templates for that service. An ODL specification is platform- and implementation language independent.

We are using the ODL specifications to express our properties on them. Since the ODL specifications are platform- and implementation language independent, the property specification also exposes this characteristic.

Most of the ideas described in this paper are also applicable to systems that are based on an IDL (Interface Definition Language) specification. In that sense, our approach can be helpful for validation of distributed services in general.

If the service developer or the service tester wants to check for violations of a property using our approach, he must first formally specify the property. The code necessary for run-time observation and validation will be generated automatically. At run-time, the events that form the property will constantly be observed and it will be checked if the property was violated. In the case of a property violation, a notification will be given. Where, when and how the run-time observation and -validation of the property is done, will be transparent to the person specifying the property.

The properties that we express should be satisfied by *every* implementation based on the given ODL specification. When expressing the properties, we do not need to worry (we do not even need to know) in which language the service will be implemented and on which platform it will run.

Figure 1 depicts the service development process in the ErnesTINA project. As soon as the ODL specifications are ready we can express properties on those specifications. We elaborate on the property specifications in section 3. Based

*This research is being partially supported by Swisscom.

0-7803-4788-9/98/$10.00 © 1998 IEEE.

on the ODL specification, a code generation tool will generate some generic observation- and validation code which needs to be linked to the actual implementation code. Details about the validation are given in section 4. When running the service, we can feed the on-line validator with our property specification. Depending on the property, observation mechanisms will be activated such that the validator will be notified about all the events that might possibly constitute a property violation.

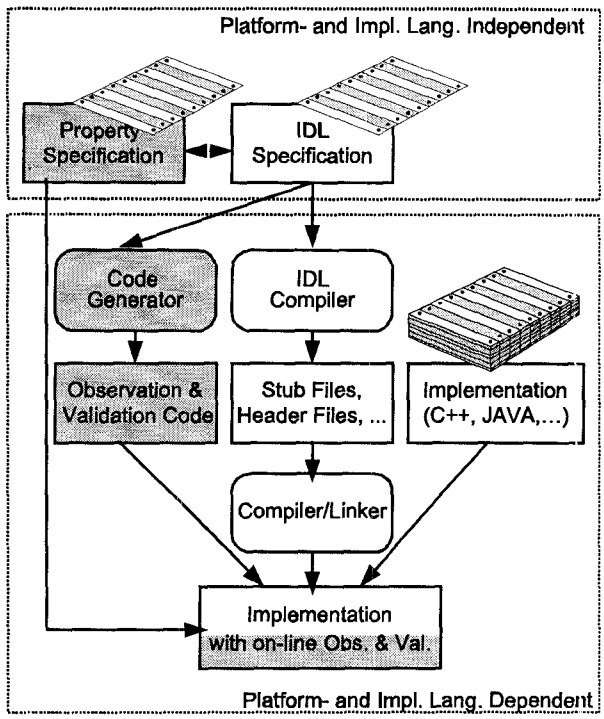

Figure 1: The Framework

Only the property specification has to be derived manually. The generation of the observation- and validation code, the examination of the observation messages and the property checking will be done automatically.

Just as the DPE (Distributed Processing Environment) and the ODL compiler hide distribution issues, the code generator and the on-line validator hide the validation problems.

When expressing properties it is *not* necessary to give a complete and formal behavior specification -this does not preclude the use of such specifications if they exist. This contrasts with the approach described in [16] where ODL is extended with a formal behavior description. A very important advantage of our property oriented approach is that it allows us to concentrate on a selected set of properties that we do not want to be violated without requiring us to give the complete behavior specification of the system. An important point, currently *not* covered in the project, is the selection of test scenarios. In the future we might also look into the possibility of automatically generating test scenarios based on the property specifications.

3 The Properties

For expressing properties we provide a set of about twenty *predefined* events that is appropriate to model industrial–strength object–oriented systems. This set has been determined by collaborating with several industrial players and by taking into account the tradeoffs between flexibility and complexity of the model and the property language. Behavior constraints are to be expressed by using these predefined events. A detailed description of all events can be found in [3]. The set of events is chosen very carefully, making it often possible to perform source code annotation for event generation in an *automatic* manner.

We consider observable events at four different levels: at the object level (e.g., the event denoting the invocation or termination of an operation on an object), the thread level (e.g., the assignment of a thread to an operation request), the process level (e.g., the arrival of an operation request at a process) and the system level (e.g., the creation of a process). Objects, threads and processes can be dynamically created and deleted.

We use linear–time temporal logic to establish a temporal relationship between events in the system.

By taking linear–time temporal logic for the specification of behavior, we can benefit from the well–known solutions for constructing test oracles.

4 Validation

Most validation[1] methods consist in either proving some sort of equivalence between the implemented system and one of its specifications. Despite the obvious advantage of capturing the requirements in a formal way, most methods we have seen used in the case of telecommunications systems (by their nature complicated and extensive) seem to suffer from a lack of efficiency:

1. These methods require an extra language. Usually the designer of the service has to specify the system twice: first with the use of the standard methodology of design and second with the use of a formal method based on a formal language such as SDL, LOTOS, etc.

2. The validation is usually based on state space exploration. As telecommunications services are enormous in the number of states to which they can evolve, state space exploration becomes extremely difficult due to state space explosion[2].

3. The validation of a formal specification does not guarantee an error-free implementation of the system. Even if the formal specification of the system has been validated, some errors can always be made when implementing the system. This problem is accentuated if the formal description of the system is not directly used as a basis for the implementation step.

The validation of properties is done at the implementation level, by observing the system and retrieving the necessary information. This information is then used to check the different properties on-the-fly. No state space exploration is necessary to validate the system and therefore, therefore the state space explosion problem is minimized. When a property is violated an error is encountered.

Taking into account that we do not have to predict the output sequence of the system (as in testing), it is possible to introduce random inputs to our system and check if the properties are violated. Similar on-line validation approaches have been undertaken by Etique [5] [4] for the

[1] "Validation is the process of determining the degree to which the requirements design, or implementation of a model are a realization of selected aspects of the system being modeled" [7]

[2] For instance, the specification of the Call Forwarding service for the Intelligent Network brings up more then 10^7 states [5]

Intelligent Network and Jard [8] for the dynamic verification of protocols.

Since the emergence of value-added services in telecommunication systems, the detection of *Feature Interactions*[3] has become a crucial problem. The validation method presented in this paper can be used to detect feature interaction without needing additional specifications. If a property upon a service has been verified when the service was working alone, this property can be violated when the service is sharing its environment with other services. By detecting this violation a feature interaction has been detected.

Section 4.1 presents the concepts of observation and spying of a distributed system. In section 4.2, the validation of the properties defined in Section 3 is explained.

4.1 Observing the Distributed System

Distributed platforms based on CORBA [15] (Common Object Request Broker Architecture) are under investigation to be used as the TINA DPE. Among these distributed platforms, many offer support for run-time observation. For example, Orbix from IONA [13] provides the *filtering* mechanism. A filter allows a programmer to specify that additional code is to be executed before or after the normal code of an operation. The CHORUS Cool distributed platform offers a mechanism similar to Orbix filters, termed *interceptors*. In our work we make the assumption that a run time observation mechanism, that we term "Filter", is provided by the distributed platform.[4]

Based on the Filters, there are different approaches to achieving run-time observation for specific properties:

- Based on the property specifications, the Filters are only put on the objects and/or processes needed to check the predefined properties. This is very efficient, but this approach lacks flexibility. Specifically, it requires that the code be recompiled each time the property changes.
- Filters are put everywhere (around every object and every process) and all the information passing through those filters is collected. The relevant events are masked out and the unneeded information discarded. This approach creates a huge overhead and heavily influences the system but gives us a comprehensive idea about what is going on.
- Filters are put everywhere but activated and parameterized as needed. This would allow feeding the validator with properties at run-time without the need to recompile the program.

Our approach is based on the last item since it gives the best compromise between flexibility and efficiency. The filters are activated and parameterized in the system based on the ODL specification and the properties. For instance, in the ODL specification we find all the operations of an interface associated with an object and having this information it is easy to construct a "programmable" filter for that object which will be able to select and to spy a specific operation. By activating and parameterizing filters, scope validation can be performed. The focus can be put only on one object of the system in order to verify local properties of this object. Additionally, global properties can be defined and validated. This hierarchical validation allows us to validate large scale systems.

Having the filter, it is then easy to send notifications containing the observed information to the validator.

Such observation mechanism can also be used for other purpose than validation: e.g., management [2].

4.2 Validating the Properties

Taking the last item of observation possibilities as a basis we have the following steps for on-line validation as depicted in Figure 2.

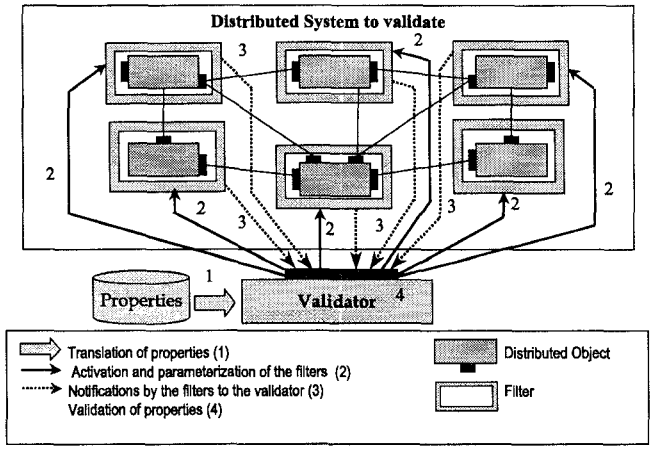

Figure 2: Validation process

1. The formalized properties are first given to the validator. The properties are interpreted and transformed into executable code via the transformation onto automata. The syntax of the properties is not detailed in this paper but is mainly following Linear Temporal Logic (LTL) [10], and tools are available to translate LTL formula into Finite State Machines (FSM). Then FSMs are transformed into executable code. The interested reader can refer to [6] or [11].
2. The validator identifies the filters which need to be activated and parameterizes them, and sends an activation message to those filters. That allows us to keep the amount of validation-related traffic in the system as low as possible.
3. The filters deliver the requested information to the validator. Each time there is an event in the system that is relevant for the property that we want to validate, the validator is notified. In some cases it is necessary to establish temporal relations between the incoming events. Therefore the validator and the filters use some time-stamp method (e.g., [14]).
4. Based on the gathered information the validator executes the code representing a FSM and is able to say if a property was violated or not. If there is a property violation, an error message will be displayed[5].

[3] "Feature Interaction are understood to be all interactions that interfere with the desired operation of the feature and that occur between a feature and its environment, including other features or other instances of the same feature" [1]

[4] This assumption is not restrictive. In the case that such a mechanism is not offered by the distributed platform, a proxy object can be added for each object within the system playing the role of the Filter.

[5] It is not yet in the scope of ErnesTINA to automatically handle some actions in the case of errors detection.

5 Case Study

The validation method proposed in the ErnesTINA project is currently applied in the framework of the SPOT (Service Pilot On TINA) project [12]. The main focus of the SPOT project are service integration and service platform interworking. The different partners of the UNISOURCE consortium and number of Telecommunications companies (Alcatel Telecom, Ericsson) will deploy different high performance multimedia TINA services over the European ATM network and ensure their interoperability. In this framework Swisscom provides a Desktop Video Conferencing Service (DVC). Our validation methodology will be applied to the DVC service.

In order to facilitate the integration of our works in the SPOT platform, we are currently working on a prototype implementing the logic of a simple basic TINA service (Basic Call). This implementation is using the Object Request Broker (ORB) from Orbix IONA [13] as a DPE. The Validator introduced in the distributed system is divided into four parts. The *Observation Manager* is in charge of getting the different notifications from the filters and restoring the causal order of the notification in the case of ordering problem. The *Properties Manager* handles the information retrieved by the Observation Manager and gives them to the Properties Checker. The *Properties Translator* transforms the properties expressed with formal syntax into executable code and gives them to the Properties Checker. The *Properties Checker* is the heart of the validation process since it checks for violation of the properties. It uses the information given by the Properties Manager and signals each property violation.

In order to spy on the distributed system, we are using the *filtering* mechanism provided by Orbix, which is of two forms: *per-process* and *per-object* filtering.

Per-process filters monitor all incoming and out-going operations to and from an address space; per-process filtering is applied when an invocation leaves or arrives at an address space. Per-object filters apply to individual objects.

The observation is based on four forms of per process filters: *out request, in request, out reply* and *in reply*. The four different filters are respectively executed: before the invocation has been transmitted, before the operation has been sent to the target object, after the operation call has been processed and after the operation response has arrived at the caller's address space.

In these filters, we specify that additional code has to be executed at each event. An event corresponds to one of the four points mentioned above where process filters can be activated. At each event (each filtering point) a notification is sent to the Observation Manager of the Validator. The notification is a "one-way" operation invoked on the Observation Manager. The information carried by these notifications is presented in Table 1.

Parameters	Information
`ProcId`	Identification of the process
`OpName`	Name of the operation invoked
`OpParameters`	All parameters of the operation
`OrigProcId`	ProcId of the initiator of the op.
`EventId`	Filter identification
`Timestamp`	Time-stamp of the process

Table 1: Parameters of the Notifications

The filtering mechanism offers the possibility of piggy-backing information with the requests. For example, the 'out request filter' can add extra data to the request and this data is removed by the 'in request filter'. We are using this mechanism to piggy-back useful information such as the identity of the process sending the request and to propagate the time-stamps stored at each process. These time-stamps are handle by the per-process filters of each process.

Although we have not yet implemented the entire validator, the current version is able to check properties in LTL [10], such as: $\Box p$, $p \Rightarrow \Diamond q$, $p \Rightarrow q \,\mathcal{U}\, r$. Such properties can be defined thanks to a GUI. In Figure 3, we present the GUI of the prototype we have implemented. Our prototype allows to monitor all the events occurring in the system and to display them in a timeline diagram and/or a table. A special window is used to specify the properties. The list of properties is shown in a window with the option to select them for on-line validation.

After having implemented a more complicate TINA service and a whole Validator, we intend to measure the performance of our system and quantify the distortion introduced by the observation mechanism added in the system.

6 Conclusions and Further Research

In this paper we have described an approach for the validation of TINA services. Our ambition in the ErnesTINA project was to produce a *practically* feasible framework for the validation of industrial strength TINA services.

Properties that are expressed independent of the actual implementation language, are verified at run-time. The service validation is transparent to the user (program developer, tester). It suffices to specify the property; an on-line observer/validator will use this property specification to check at run-time that it is not violated. The generation of the observation- and validation code will be handled by a tool.

We have implemented a first prototype that is able to verify simple properties. The prototype is currently being extended to handle more sophisticated properties.

The concepts, ideas and tools that are currently being developed in the framework of the ErnesTINA project will be applied to an industrial-strength TINA service, i.e., the desktop video-conferencing system of the SPOT project.

7 Acknowledgements

The authors would like to thank Swisscom for many interesting discussions. We thank S. Koppenhoefer and D. Hutchinson for their useful and constructive remarks. We are grateful to H. Karamyan for his work on the prototype implementation.

References

[1] E.J. Cameron, N.D. Griffeth, Y.-J. Lin, M.E. Nilson, W.K. Schnure, and H. Velthuijsen. A feature interaction benchmark for IN and beyond. In *Feature Interactions in Telecommuications Systems*, pages 1–23. Bouma, L.G and Velthuijsen, H., Amsterdam, IOS press edition, May 1994.

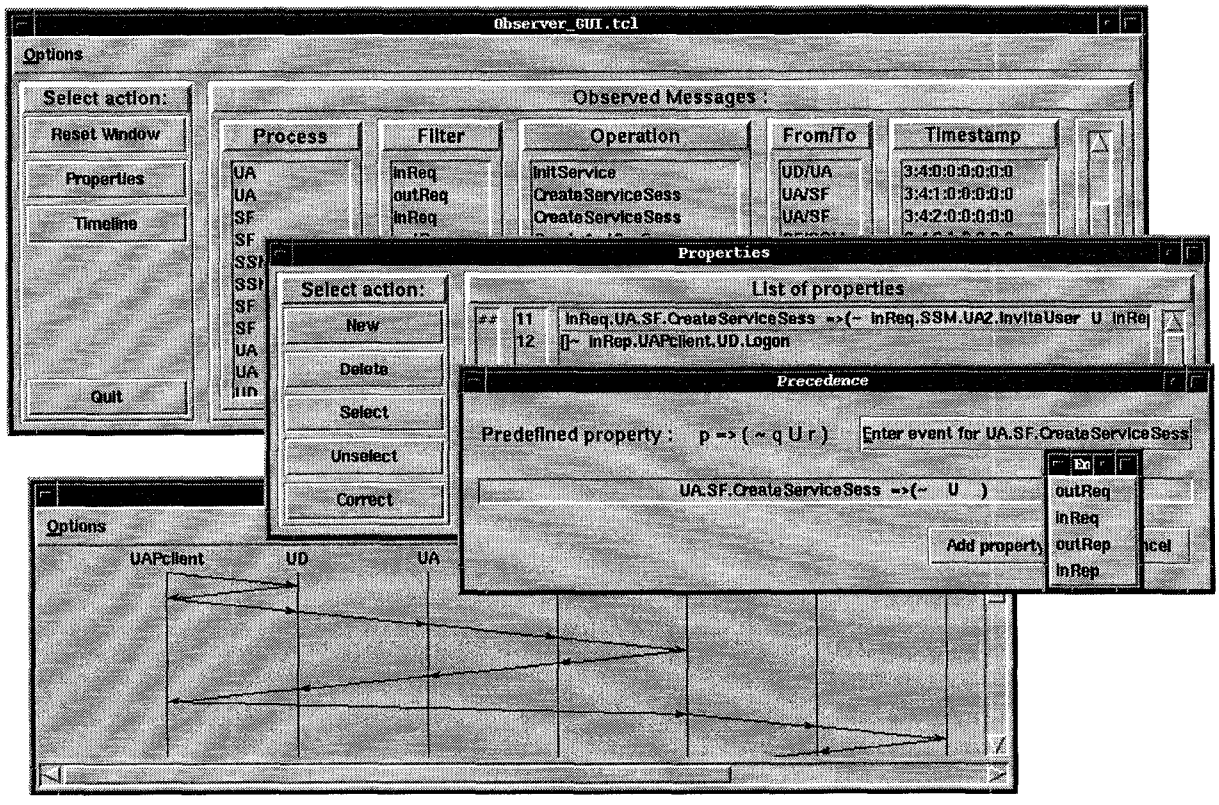

Figure 3: screen dump of the prototype

[2] J. A. G. de Queiroz and E. R. M Madeira. Management of corba object monitoring for multiware platform. In *Open Distributed Processing and Distributes Platforms*, pages 122–133. Chapman & Hall, 1997.

[3] F. Dietrich, X. Logean, S. Koppenhoefer, and J.-P. Hubaux. A temporal logic–based approach to the design of object–oriented distributed systems. Technical report, Swiss Federal Institute of Technology, Lausanne, 1998. Available from the authors.

[4] P.-A. Etique, J.-P. Gaspoz, J.-P. Hubaux, et al. Validation of an object-oriented service specification for the Intelligent Network. In *TINA'95, Integrating Telecommunications and Distributed Computing - from Concepts to Reality.*, volume 2, pages 561–576, Melbourne, Australia, 1995.

[5] Pierre-Alain Etique. *Service Specification, Validation and Verification for the Intelligent Network.* PhD thesis, Swiss Federal Institute of Technology Lausanne, Telecommunications Laboratory, 1995.

[6] Pascal Grimont and Pierre Wolper. *From Modal Logic to Deductive Database*, chapter 4 Temporal Logic, pages 165–233. Wiley, 1989.

[7] *The New IEEE Standard Dictionary of Electrical and Electronics terms*, volume 1. Chistopher J.Booth, Ed., 5 edition, 1993.

[8] Claude Jard. *Vérification dynamique des protocoles, documents d'habilitation.* IRISA + IFSIC, 1994.

[9] B. Kitson, P. Leydekkers, N. Mercouroff, and F. Ruano. *TINA Object Definition Language (TINA-ODL) Manual.* TINA-C, June 1995. Version 1.3.

[10] Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety.* Springer-Verlag, 1995.

[11] H. McGuire. *Two Methods for Checking Formulas of Temporal Logic.* PhD thesis, Department of Computer Science, Stanford University, Stanford, California, 1995.

[12] N Mercouroff, J. Bengtsson, P. Hellemans, and L. Lehmann. Implementation of sevices for computer supported cooperative work on TINA : the SPOT project. submitted to ISS Conference, 1997.

[13] *Orbix 2, Distributed Object Technology.* IONA Technologies Ltd., 1996.

[14] A. Schiper, J. Eggli, and A. Sandoz. A new algorithm to implement causal ordering. In *Third Int'l Workshop Distributed Algortithm*, pages 219–232, Berlin, 1989.

[15] Jon Siegel. *CORBA, Fundamentals and Programming.* John Wiley & Sons, Inc., 1996.

[16] B. Stepien, K. Farooqui, and L. Logrippo. An experience modelling telecommunications systems using odp-dlcomp. In Stefani JB. Najm E., editor, *Formal Methods for Object-based Distributed Systems*, London, 1996. Chapman & Hall.