



Evolution Management and Process for
Real-Time Embedded Software Systems

The Interface Wrapper Architecture

Deliverable D.2.1-D.2.2 Appendix C

Written and Edited by Stefan Van Baelen, K.U.Leuven

15 December 2003

Version 1.0

Status final

Public Version



I T E A

INFORMATION TECHNOLOGY

FOR EUROPEAN ADVANCEMENT

This document is part of the work of the EUREKA Σ! 2023 – ITEA 00103 project EMPRESS.
Copyright © 2002-2003 EMPRESS consortium.

Authors/Partners:

Partner	Author	Email
K.U. Leuven	Stefan Van Baelen	Stefan.VanBaelen@cs.kuleuven.ac.be

Document History:

Date	Version	Editor	Description
15 Dec 2003	1.0	Stefan Van Baelen	Public version based on internal version 2.4

Filename: D2.1_D2.2_v1.0_Appendix_C_Public_Version.doc

Abstract

Since components are loosely coupled exchangeable and reusable software elements, their interaction should be well defined. Although interfaces clearly describe the interaction of a component with all other components in the system, they can impose a burden on the evolution of connected components. Components can and should be able to evolve individually, the provided and required interfaces of components involved in a connection will sooner or later diverge. Such interface changes often cause an exponential ripple effect, since components can be connected to a significant number of other components. A solution should be provided to enable interfaces to evolve while maintaining the consistency of the defined connections, at least as long the interfaces remains compatible.

This document describes the interface wrapper architecture, which is a generally applicable architecture that can provide support to component and interface evolution, diminishing the potential exponential ripple effects of such changes. Interface wrappers are able to solve map incompatibilities, such as those between message names, parameters, return values, synchronous versus asynchronous messages and message order. Both message and wrapper will contain the necessary version transformation logic to solve incompatibilities. As such, interface wrappers enable evolution of interfaces and provide the necessary message transformations.

Keywords:

Component Architecture, Component Adaptor, Component Wrapper, Architectural Support.

Table of Contents

ABSTRACT	3
TABLE OF CONTENTS.....	4
1 INTRODUCTION.....	5
1.1 REUSING OBJECTS AND COMPONENTS	5
1.2 THE INTERFACE EVOLUTION PROBLEM	5
2 GLUE COMPONENTS.....	6
2.1 GOAL OF GLUE COMPONENTS	6
2.2 EXISTING PROBLEMS WITH GLUE COMPONENTS	7
3 INTERFACE WRAPPERS.....	9
3.1 GOAL OF INTERFACE WRAPPERS	9
3.2 ARCHITECTURE OF INTERFACE WRAPPERS.....	10
3.3 BENEFITS OF INTERFACE WRAPPERS	11
3.4 ADDITIONAL FUNCTIONALITY WITHIN INTERFACE WRAPPERS.....	12
4 CONCLUSION	13

1 Introduction

1.1 Reusing Objects and Components

Due to the amount of software systems to be developed and code to be written, reuse emerged as one of the critical success factors for the future of software development. Object-orientation was introduced as one of the first steps to a radical change in software development, tackling the problem of reuse by introducing objects as the core elements in a software system. However, classes and objects seemed to be too fine-grained to serve as the main item to be reused. In addition, objects contained too much dependency to other objects to be effective as the level of proper reuse.

Component technology introduced an alternative solution for the problem of reuse. Components were introduced as loosely coupled, exchangeable and reusable software elements that were of the granularity to support software reuse. Although components are rather independent software elements, their interaction should be well defined. Therefore, the interaction between components has been defined by a set of interfaces to provide access to the services of a component (provided interfaces). Components also define the set of interfaces to the services they expect from other components in the system (required interfaces). As such, component compositions can be made by connecting (wiring) a provided interface of a component to a required interface of another component. This is illustrated in Figure 1, in which the ITEA-DESS [DES01] component notation is used.

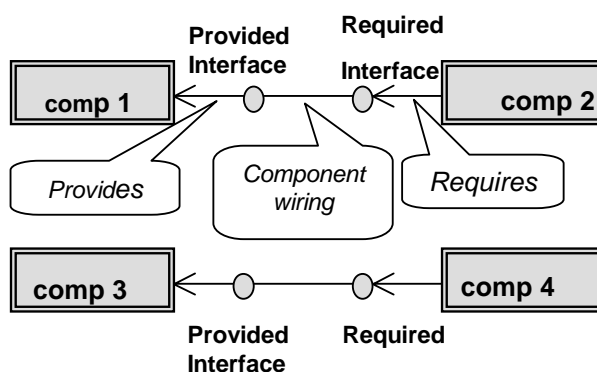


Figure 1: Component interfaces and connections

Components offer the ability of isolating certain parts of an application, partitioning it into a number of independent modules that are loosely coupled with each other.

1.2 The Interface Evolution Problem

2 Glue Components

Although interfaces clearly describe the interaction of a component with all other components in the system, they can impose a burden on the evolution of connected components. Since components can and should be able to evolve individually, the provided and required interfaces of components involved in a connection will sooner or later diverge, as illustrated in Figure 2. Such interface changes often cause an exponential ripple effect, since components can be connected to a significant number of other components. A solution should be provided to enable interfaces to evolve while maintaining the consistency of the defined connections, at least as long the new interface versions remain compatible to the old ones. Compatibility of interfaces is hereby defined as the ability of mapping all operations and events of the required interface onto the provided interface, while eventually transforming message names, signatures and parameter data, introducing default values for missing parameters, injecting messages and/or buffering and changing the order of messages.

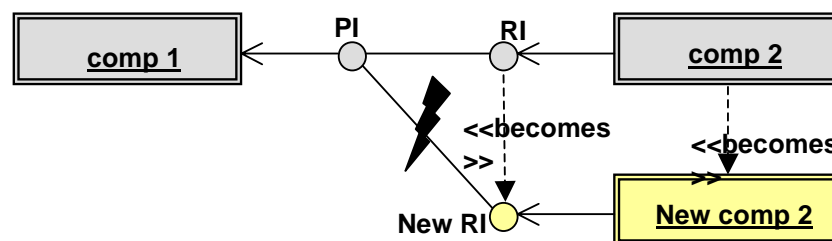


Figure 2: The interface evolution problem

The goal of the interface wrapper architecture that is presented in this document is to define a generally applicable architecture that can provide support to component and interface evolution, diminishing the potential exponential ripple effects of such changes.

2 Glue Components

2.1 Goal of Glue Components

The classical solution to solve incompatibilities between interfaces is to introduce glue components, or so-called adaptors, between components that must be connected. These glue components perform the transformation between incompatible required and provided interfaces of the two components. When a provided interface evolves into a new version that is incompatible with the previous ones, a glue component will be introduced transforming messages according to the old version of the interface into messages of the new interface version. This is illustrated in Figure 3.

Whether or not a single glue component can transform the messages from all sending components, or whether a glue component should be introduced for each sending component, is dependent on the nature of the interface change and the complexity of the transformation to be performed.

2 Glue Components

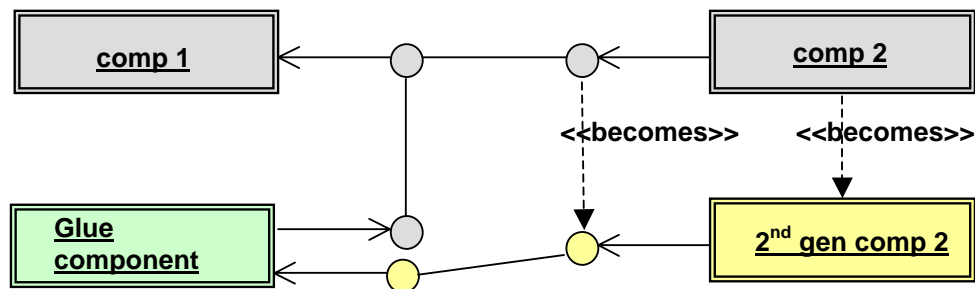


Figure 3: Glue Component Usage

2.2 Existing Problems with Glue Components

Using glue components to cope with interface evolution does not suffice for two major reasons:

- Glue components only support the transformation of a pair of incompatible interfaces, transforming a specific version of a required interface into another version of a provided interface. When both interfaces can evolve after an amount of time, at least three glue components are needed:
 - One transforming the old version of the required interface into the new version of the provided interface
 - One transforming the new version of the required interface into the old version of the provided interface
 - One transforming the new version of the required interface into the new version of the provided interface

This is illustrated in Figure 4.

- When an interface evolves into a third generation version, the second generation glue component should be replaced with a third generation glue component as presented in Figure 5, or two cascaded glue components should be introduced transforming the third generation interface into a second generation interface, which at its turn has to be transformed into a first generation interface, as presented in Figure 6.

2 Glue Components

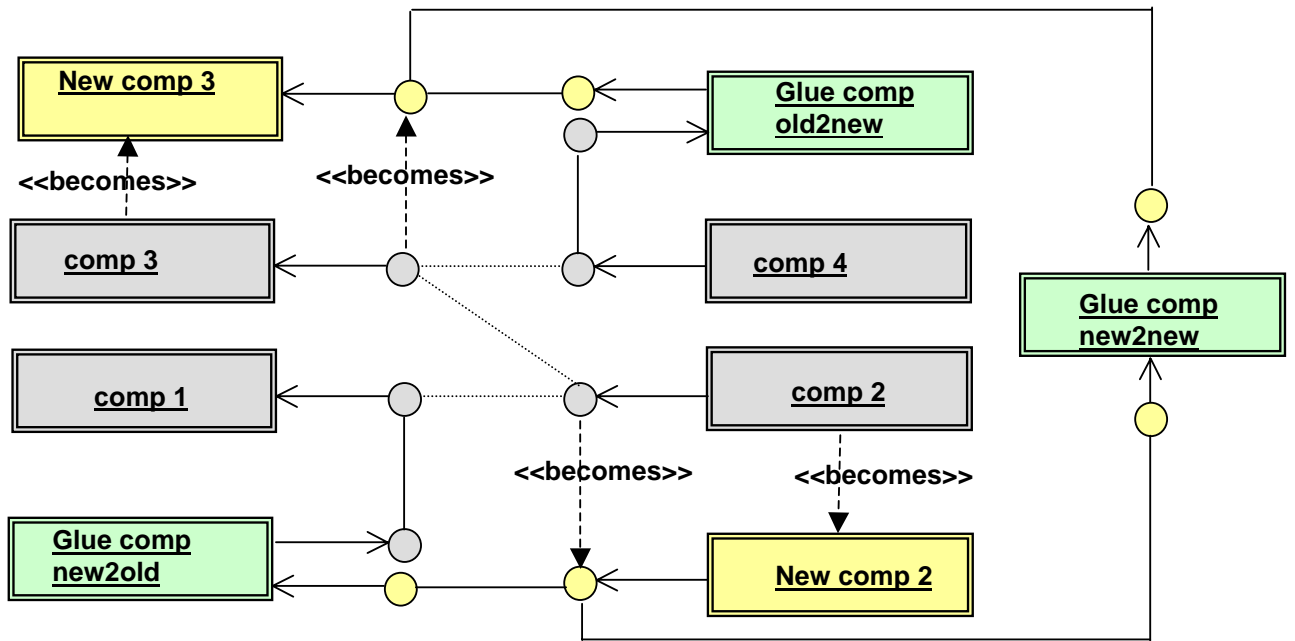


Figure 4: Pair-wise glue component explosion

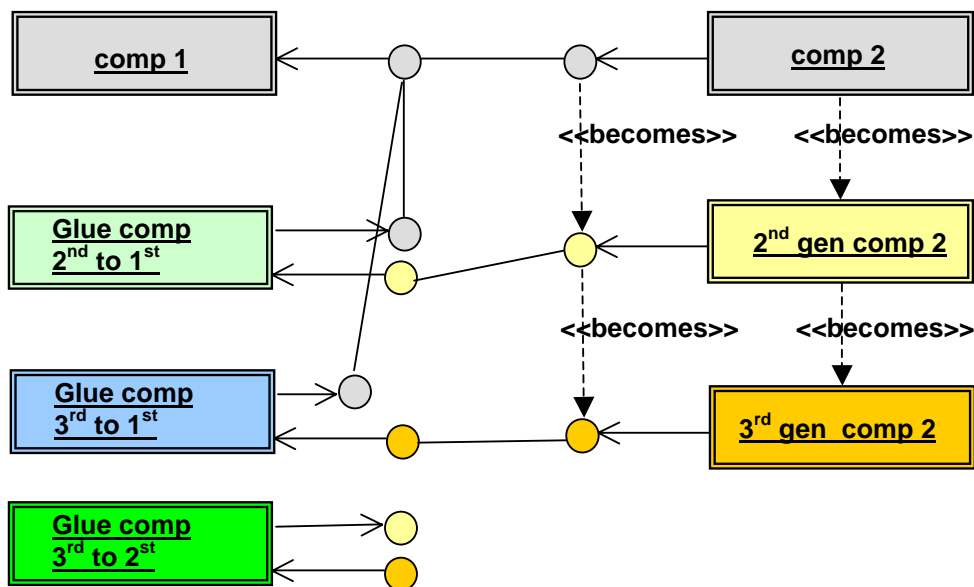


Figure 5: Combinatorial explosion of Glue Components

3 Interface Wrappers

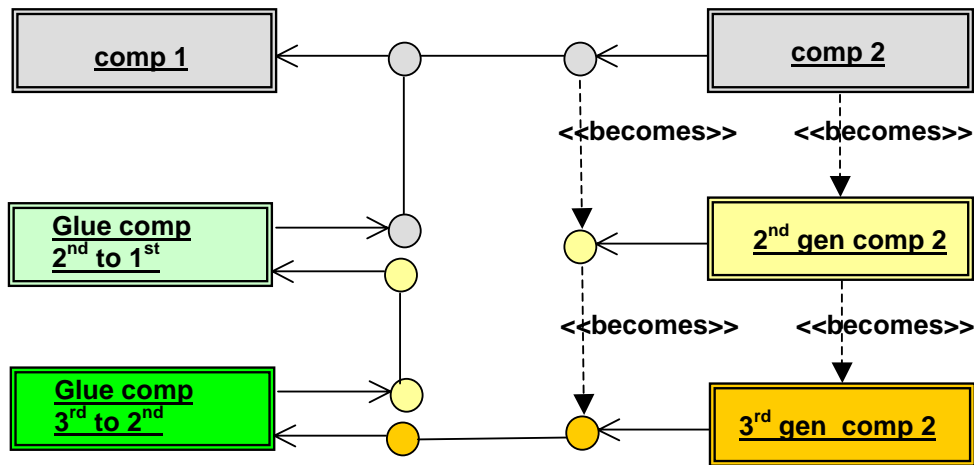


Figure 6: Cascading Glue Components

3 Interface Wrappers

3.1 Goal of Interface Wrappers

Interface wrappers aim to provide an adequate solution for most of the problems of interface evolution. By standardising inter-component communication, all interface wrappers will be able to exchange messages regardless of the changes that occurred to the component interfaces. The problem of interface evolution will be shifted to an interface-mapping problem, whereby each interface version must be able to map incoming and outgoing message between the specific interface and the standardized interface.

Interface wrappers play the role of message transformers, which are applied each time a message is sent or received through the interface. As such, each special-purpose interface message is transferred into a general-purpose inter-component message containing all relevant information and vice versa, as presented in Figure 7.

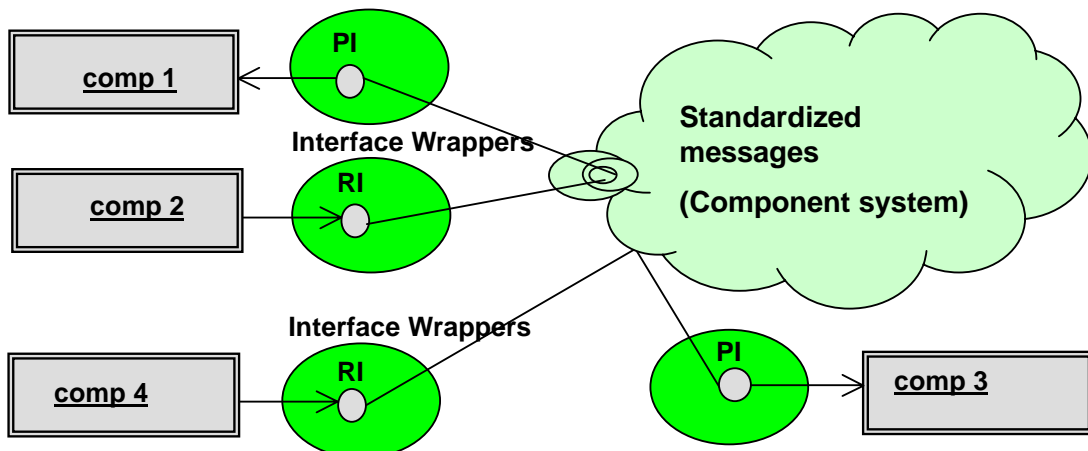


Figure 7: Interface wrappers usage

3 Interface Wrappers

The standardized message is then being transferred to the interface wrapper of the receiving component using the services of a component system or communication middleware. The interface wrapper of the receiving component will then reconstruct the special-purpose interface message according to the receiving interface.

3.2 Architecture of Interface Wrappers

The interface wrapper architecture consists of the following elements:

- A standardized asynchronous message exchange protocol defined by the interface wrapper architecture and supported by a component system or communication middleware. Every message, return value and raised exception will be passed by a standardized message containing all relevant information in a hash table (interaction id, message name, parameters values, return values, communication parameters, ...)
- An interface wrapper layer intercepting all outgoing messages, transforming it into a standardized component system message and attaching the necessary interaction identification number and communication parameters. The component system will deliver this component message to the appropriate interface wrapper layer, which will decompose the received generic message, extract the relevant information from it and reconstruct the original message that will be delivered to the receiving component (using reflection or through a direct method call). When the component responds with a return value or raises an exception, the interface wrapper layer will use the information to compose a generic message containing the response information, while adding the corresponding interaction identification number. The interface wrapper layer of the sending component receives this generic message and will reconstruct the message response and deliver it to the sending component.
- The components themselves will not be affected by the interface wrapper layer. They will communicate without having any knowledge of the in-between interface wrapper layers and component system. They only reference to the corresponding interface wrapper layers and communicate with it as if they were directly communication to the target component (receiving component in case of a required interface and sending component in case of a provided interface).

The detailed structure of the interface wrapper architecture is presented in Figure 8. A dedicated wrapper component will be attached to each interface, transforming the specific interface to the standardized interface defined by the component system. The standardized interface only offers nameless message passing, whereby a hash table of key-value pairs is attached to the message.

3 Interface Wrappers

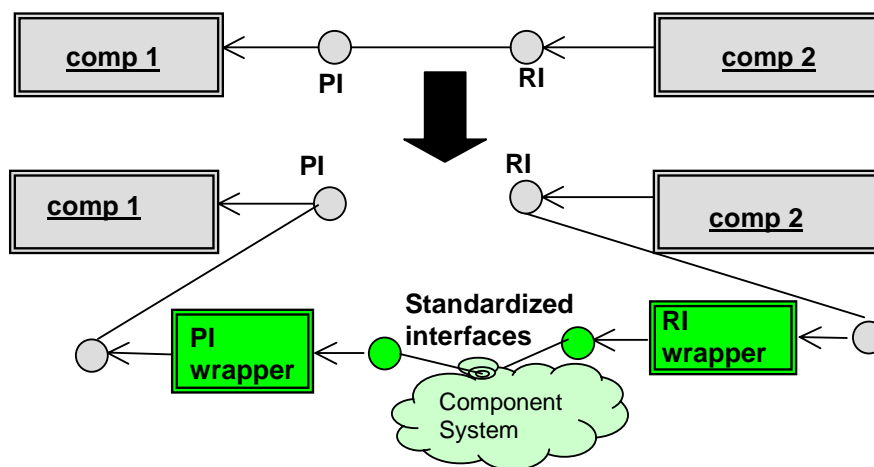


Figure 8: Realization of the interface wrapper architecture

Interface wrappers can be generated automatically, based on the description of the interface they have to wrap and the general-purpose inter-component message protocol defined by the component system. Based on the interface, all relevant information can be extracted and the appropriate hash table with key-value mappings can be generated. For incoming messages, the information can be retrieved out of the hash table, using default or dummy values when the information is not been included in the hash table.

3.3 Benefits of Interface Wrappers

Interface wrappers enable evolution of interfaces and provide the necessary message transformations in order to provide communication between components using different versions of an interface. Interface wrappers can map incompatibilities, for instance between messages names, parameters, return values, sync/async messages and message order.

- Both message and wrapper will contain version transformation logic, based on the greatest common factor between the interface wrappers. This is illustrated in Figure 9.
 - The outgoing interface wrapper will always send a message according to its latest version. So if the interface of the sending component is of the X^{th} generation, the message will contain the information according to the X^{th} version. It is also responsible to include all information that will be needed to downgrade the message to a lower version. If this includes transforming parameter information, it should also include this transformed information, or provide the functionality necessary to perform such transformation (message transformer).
 - When the receiving component supports a higher version of the interface, the interface wrapper must be able to upgrade an older message to the newest interface version. So if the receiving component is of the Y^{th} generation with $Y > X$, then the receiving component should be able to upgrade the message

3 Interface Wrappers

version X into a message version Y. When $Y < X$, the receiving interface wrapper can use the message transformer provided by the sender to downgrade the message according to the appropriate version level.

Using this technique, only one wrapper per interface version is needed. The responsibilities are shared between the sending component (responsible for downgrading a message) and the receiving component (responsible for upgrading a message)

- The interaction between synchronous and asynchronous message-based interfaces can be mapped as follows: For synchronous outgoing messages, the interface wrapper will perform a simulation by blocking the component sending the message until the return message has arrived. When a message arrives at the receiving interface wrapper that must be mapped onto a synchronous component message, the receiving interface wrapper will automatically generate a new message that will contain all relevant information about the return values or thrown exceptions of the receiving component.
- In case of message order conflicts between the sending and receiving interface, the receiving interface wrapper will temporarily buffer incoming messages in order to reconstruct an appropriate message order for the receiving components. This buffering is based on information within the receiving interface wrapper (in case of interface upgrading) or within the message itself (in case of interface downgrading). The same technique is used before sending return messages.

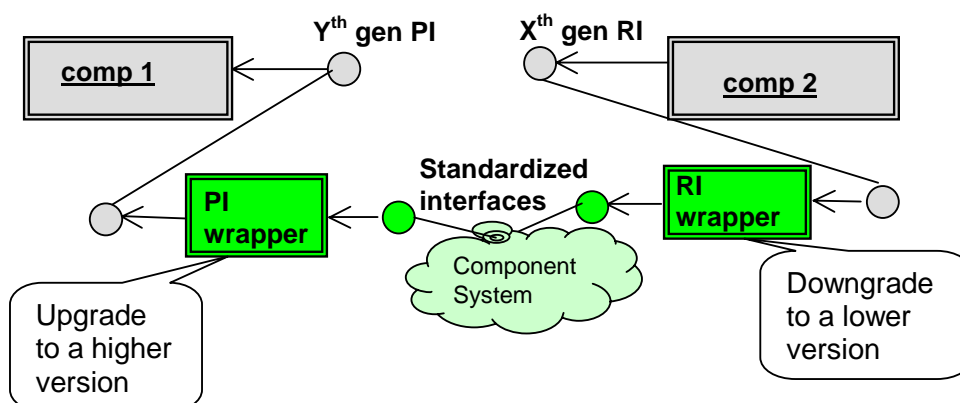


Figure 9: Version transformation logic

3.4 Additional Functionality within Interface Wrappers

In addition to the basic communication services, the interface wrappers can also offer a lot of added value:

- Component referencing issues can easily be hidden inside the interface wrapper. The choice between direct component referencing, port-based communication, component look-up using directory services or name-based component referencing can be made on the level of the component system, and encapsulated within the interface wrapper.
- Fine-tuning of the communication channels (connectors)

4 Conclusion

- Retrieval of status information about the communication and the connectors
- Logging and accounting facilities for component usage
- Security, authentication and authorization of the communication
- Additional message processing capabilities, which can be done by introducing specific filters for outgoing or incoming messages.

A lot of these additional services of the interface wrappers do not have to be directly visible to the component or its sender. The services and related information that must be available could be accessible through a dedicated service API of the interface wrapper.

4 Conclusion

Since components are loosely coupled exchangeable and reusable software elements, their interaction should be well defined. Therefore, components are defined by a set of provided and required interfaces, which define the services they offer to and need from other components. As such, component compositions can be made by connecting (wiring) a provided interface of a component to a required interface of another component.

Although interfaces clearly describe the interaction of a component with all other components in the system, they can impose a burden on the evolution of connected components. Since components can and should be able to evolve individually, the provided and required interfaces of components involved in a connection will sooner or later diverge. Such interface changes often cause an exponential ripple effect, since components can be connected to a significant number of other components. A solution should be provided to enable interfaces to evolve while maintaining the consistency of the defined connections, at least as long the interfaces remain compatible.

The interface wrapper architecture that was presented in this document provides a generally applicable architecture that can support component and interface evolution, diminishing the potential exponential ripple effects of such changes. Interface wrappers are message transformers that are applied each time a message is send or received through an interface. As such, each special-purpose interface message is transferred into a general-purpose inter-component message containing all relevant information and vice versa.

Interface wrappers are able to solve map incompatibilities, such as those between message names and signatures, parameters, return values, synchronous versus asynchronous messages, and message order. Both message and wrapper will contain the necessary version transformation logic to solve incompatibilities. This is based on the greatest common factor principle, allowing each version of the interface to have only one wrapper to communicate with all other versions. As such, interface wrappers enable evolution of interfaces and provide the necessary message transformations.