# C3DS Design and Development Methodology

Edited by Valérie Issarny
Solidor Research Group
INRIA-Rennes and INRIA-Rocquencourt
Email: Valerie.Issarny@inria.fr

**Abstract**

The C3DS project aims at easing the design and implementation of complex services out of existing ones. Towards that goal, the project is designing and prototyping the TCCS environment, which decomposes into: (i) the TCCS development environment for the design and implementation of complex services based on the combined specifications of software architecture and workflow schema, and (ii) the TCCS platform for the execution of complex services, which is a middleware platform that builds on the CORBA standard and offers middleware services for the execution of software agents and workflows. In this report, we detail the C3DS design and development methodology, that is the development process for complex service provisioning using the overall TCCS environment. Specifically, we concentrate on the process phases that relate to exploiting the TCCS environment and hence do not consider complementary phases such as those appertained to requirement analysis and testing. The overall development process as addressed in the C3DS project comprises the phases for the design, implementation, and deployment of complex services, which come along with associated tool support.

## 1 Introduction

The C3DS project targets the provisioning of complex services out of existing ones. In other words, the project aims at providing means for the easy composition of existing services. Thus, the resulting TCCS environment that is being developed in the C3DS project for complex service provisioning naturally builds upon the following technologies:

- *Software architecture*, which has been proven successful for the design, analysis and implementation of complex software systems by enabling their abstract description in terms of the composition of components via connectors, using an *Architecture Description Language* (ADL).

- *Workflow management systems*, which were initially introduced for the automation of business processes by supporting the composition of various activities, possibly crossing distinct organizations. In general, such systems may serve for realizing any kind of distributed processes since their primary functionality lies in the coordination of a set of inter-dependent tasks. Basically, a workflow management system offers a language for the definition of workflow schemas (or workflow scripts), and a number of runtime services for supporting the execution of those schemas.

- *Software agent systems*, which enable enriching existing software through additional computation as achieved by agents. In addition, the mobile agent technology is now being considered as a key technology for meeting the requirements imposed by the new shift in the area of distributed systems, i.e., increasing mobility and usage of the Internet [Cardelli 1998].

- *Middleware infrastructures* that provide reusable solutions to problems frequently encountered in many different types of distributed applications, e.g., heterogeneity, interoperability, security, transactions, fault tolerance, etc. We base more specifically our work on the CORBA standard that is now widely accepted.

In the above framework, complex service provisioning amounts to:

(i) Designing the *service schema* (where this term is used by analogy with the notion of *workflow schema*) and associated *service software architecture*, i.e, precisely defining the flow of computation to be achieved by the given service through the composition of base services offered by the underlying software architecture. This may be achieved in a number of ways depending on the specifics of both the schema and architecture. In the framework of C3DS, we specifically focus on service schemas that translate into workflow schemas.

(ii) Implementing the *workflow schema*, i.e. implementing the underlying service architecture, and mapping the schema onto it.

(iii) Deploying the complex service together with necessary monitoring and control support for further evolution.

The above roughly corresponds to the overall C3DS development process, which is further discussed in the next section. Let us notice here that the aforementioned development phases may be realized in a modular way in that a complex service may actually be defined through a number of service schemas, possibly defined hierarchically. In addition, all the above phases are aided through the provision of CASE tools, especially those for carrying out analyses of complex services so as to assess the undertaken development decisions. Analysis support is presented in Sections 3 to 5, which respectively detail the design, implementation, and deployment phases of our development process. Finally, we conclude in Section 6,

summarizing the C3DS design and development methodology and highlighting the associated tool support provided by the TCCS environment.

# 2 The C3DS Development Process

In this section, we give an overview of the overall C3DS development process, which relates to exploiting the TCCS environment for complex service provisioning. Hence, the proposed process is not to be considered as being complete, and ignores complementary phases such as those appertained to requirement analysis and testing, which should be addressed using existing processes. For instance, the interested reader may refer to [Baresi *et al.* 1999] and [Weske *et al.* 1999] for examples of development processes appertained to workflow applications. In addition, we provide an informal definition of the process using box-and-line diagrams. However, these are precise enough to give a clear understanding of the proposed process.

Prior to address the process definition, we first have to qualify the complex services for which the TCCS environment targets provisioning. As previously discussed in *Deliverable A1*, the C3DS project does not target a specific application domain. However, there is a number of assumptions that underly the project, and hence enable precising the kind of complex services that we consider in the framework of C3DS:

- We assume a number of provided base services for composing more complex ones. Hence, we consider that there exists a repository of base services, or at least a repository of their abstract description under the form of components[1]. Such a repository is then exploited by software designers when elaborating complex services. In the same way, for the sake of software reuse, workflow schemas that are developed are stored in the service repository.
- Complex services are inherently distributed and possibly involve distinct organizations. In particular, interactions among base services may be realized over the Internet.
- Participating organizations are TCCS-enabled, i.e., they all provide execution environments that conform to the TCCS environment.

Figure 1 depicts the resulting model of the overall C3DS development process. Phases written in italic correspond to those that we do not address and for which the use of software development methods as provided by existing processes should be investigated; other phases could further be considered based on the selected process. As depicted by dotted arrows in the figure, the undertaken development decisions may be revised at each phase, leading to

[1]Let us notice here that according to the general definition of ADLs, a component may actually be a complex one (i.e. a configuration or architecture) built out of more primitive components.

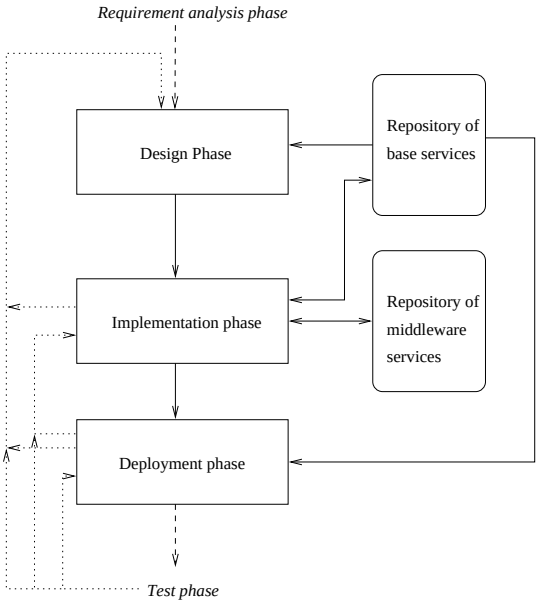



Figure 1: The C3DS development process model

return to some previous development phase. The three phases that are of interest here are the ones relating to the design, implementation, and deployment of complex services. The design phase takes as input an abstract description of the service schema to be developed, and uses the repository of available base services, which are the ones available in the given context of development. The output of this phase is the overall abstract specification of the complex service under the form of a workflow script and associated software architecture. This specification is then provided to the implementation phase. During this phase, the developer implements the components that were introduced during the design phase. In addition, this phase comprises the implementation of a customized TCCS-based middleware for the execution of the complex service according to required non-functional properties. This latter development step is aided through a CASE tool, which uses a repository of the TCCS-compliant middleware services that are available within all the participating organizations. As depicted by the double-edge arrows, repositories may be updated during the implementation phase, hence making available for future complex service provisioning, the new service together with the associated underlying middleware architecture. The result of the implementation phase is the complex service's overall software architecture that prescribes the service's configuration in terms of application-level components and complementary middleware components. The latter set of components actually embeds the ones relating to workflow execution (i.e. task and task controller objects), and those appertained to middleware customization. Finally, the service configuration is exploited by the deployment phase

for distribution of the configuration components over the sites of the participating organizations. The configuration is further enriched so as to support later evolution in response to changes in the user requirements or in environmental factors. The result of this last development process phase is a ready-to-use complex service.

**Example:** For the sake of illustration, we take the example of a business process from the e-commerce domain throughout the report [Shrivastava & Wheater 1999]. The example is based on the *order and pay* steps of a browse, order, pay then deliver style electronic commerce interaction. The order and pay steps involve four *organizations*: customer, supplier, and two banks. They further rely on the following interactions, which set the composing base services distributed over the organizations:

(1) The customer places an order with the supplier.

(2) The supplier validates the order and assuming OK calculates the value of the order.

(3) The supplier sends a quote to the customer.

(4) The customer processes the quote.

(5) The customer instructs its bank to initiate payment; the customer also confirms the order to the supplier

(6) The supplier confirms payment; if OK, it will initiate goods shipment, else the order is cancelled.

# 3 The Design Phase

Let us now concentrate on the design phase of complex service provisioning. Here, we assume that an *abstract service schema* is provided as input, resulting from the requirement analysis phase. Basically, this schema characterizes the complex service to be developed in terms of its composing base services and the temporal dependencies among them. The design phase then consists of two inter-dependent sub-phases:

- Designing the service's *structural view*, i.e, designing the components providing the base services that are to be composed.
- Designing the service's *behavioral view*, i.e., designing the workflow schema that coordinates the execution of the base services.

The two above design sub-phases are further complemented with a number of analyses so as to carefully assess the undertaken design choices. The overall design phase is depicted in Figure 2 and is further detailed below.

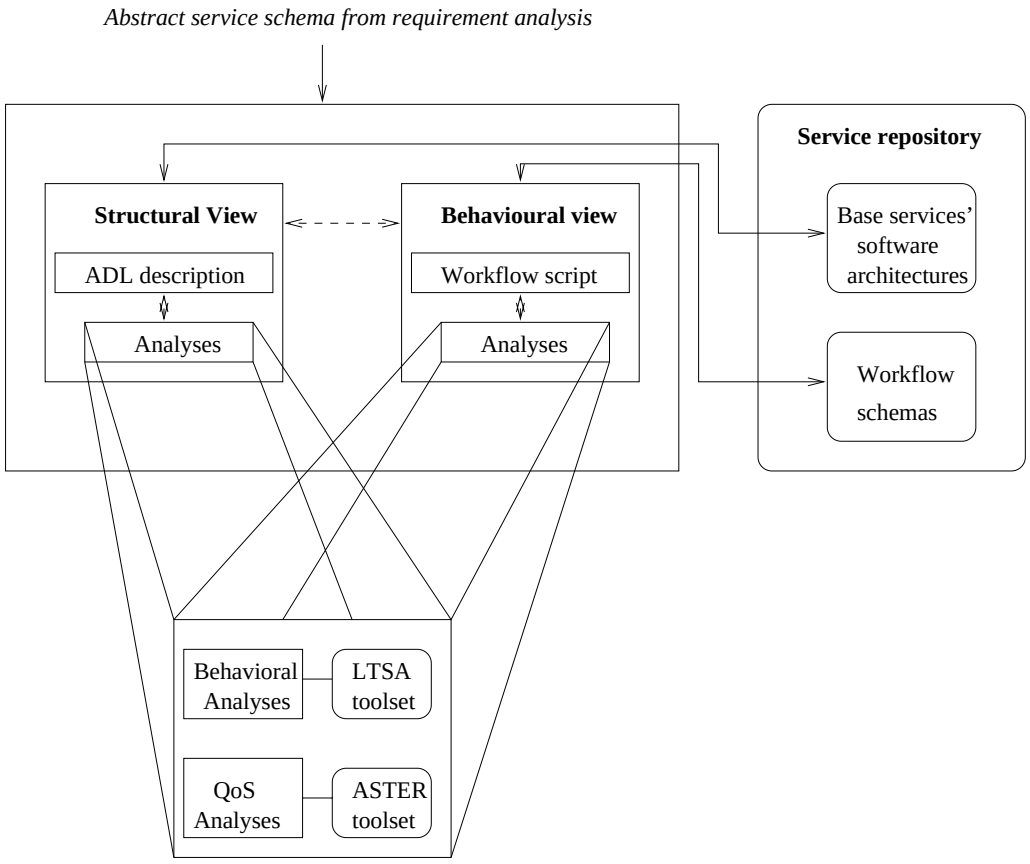



Figure 2: The C3DS design phase

## 3.1 Designing the service's structural and behavioral views

Given available base services, possibly resulting from the previous development of complex services, the design of a given service requires to elaborate the various base services to be coordinated using a workflow schema.

**Structural view:** The structural view serves characterizing the various base services to be coordinated, using software architecture description. In the simplest case, the service is already available and hence belongs to the repository of services. However, it is expected that available services will have to be enriched so as to fit the complex service's requirements as identified by the service schema. Using the TCCS environment, making evolve a given base service may be achieved in two ways through the introduction of either (i) CORBA

objects, or (ii) software agents. Then, each constituent service is precisely defined using the TCCS ADL (see *Deliverables B1.1* and *B1.3*) as a software architecture, which sets all the components (i.e. available base services and additional ones) and the interactions among them. In that context, the software architecture associated to a complex service is the set of software architectures defining the base services composing the complex service, which may possibly share common components.

**Behavioral view:** The behavioral view refines the service schema by precisely setting the workflow schema to be run so as to provide the targeted complex service. The workflow schema to be deployed characterizes the temporal and data-flow dependencies among the components specified by the complex service's structural view (or software architecture). Furthermore, this schema accounts for possible failures, and thus integrates treatment relating to the occurrence of either underlying system or application-level exceptions. The description of workflow schemas is provided using the TCCS workflow language (see *Deliverable B1.3*).

**Example:** Let us consider the example from the e-commerce domain that was introduced in the previous section. Focusing on the *order and pay* steps, the six interactions that were mentioned actually define the abstract service schema. Such a schema may be rewritten in terms of a box-and-line diagram as depicted in Figure 3. Although quite informal, the depicted schema shows that the complex service is made out of four base services, each being provided by the involved organizations. In addition, those base services are themselves complex, being composed of inner services, which may possibly be already available within the service repository. In particular, the *process_order* and *confirm_order* services are inner service schemas, the latter being nested within the former. Concentrating further on the *process_order* service, this includes the *calculate_order_value* base service, which may possibly be already offered by the supplier organization. If so, the service description belongs to the service repository and is used for further design elaboration. On the other hand, if the base service is not provided by the supplier, two cases have to be considered:

(1) The service has formerly been designed in the context of another similar organization, in which case the corresponding software architecture belongs to the service repository. Should this software architecture *match* the given supplier organization (i.e. embedded components and connectors are available), the service's design is used for further processing. If there is only a partial match (i.e. some architectural elements are missing) then the matching part of the architecture is taken as an initial design and is handled according to the alternate case given below.

(2) The service has to be designed, possibly using an initial partial architecture. In this case, the software architecture needs to be enriched through the revision of existing components, integration of new components, or a combination of both. Typically, those

components will correspond to either CORBA objects or software agents. Let us further notice that in the case of component revision, this may lead to, possibly dynamic, software reconfiguration within the given organization if the revision impacts upon existing persistent services.

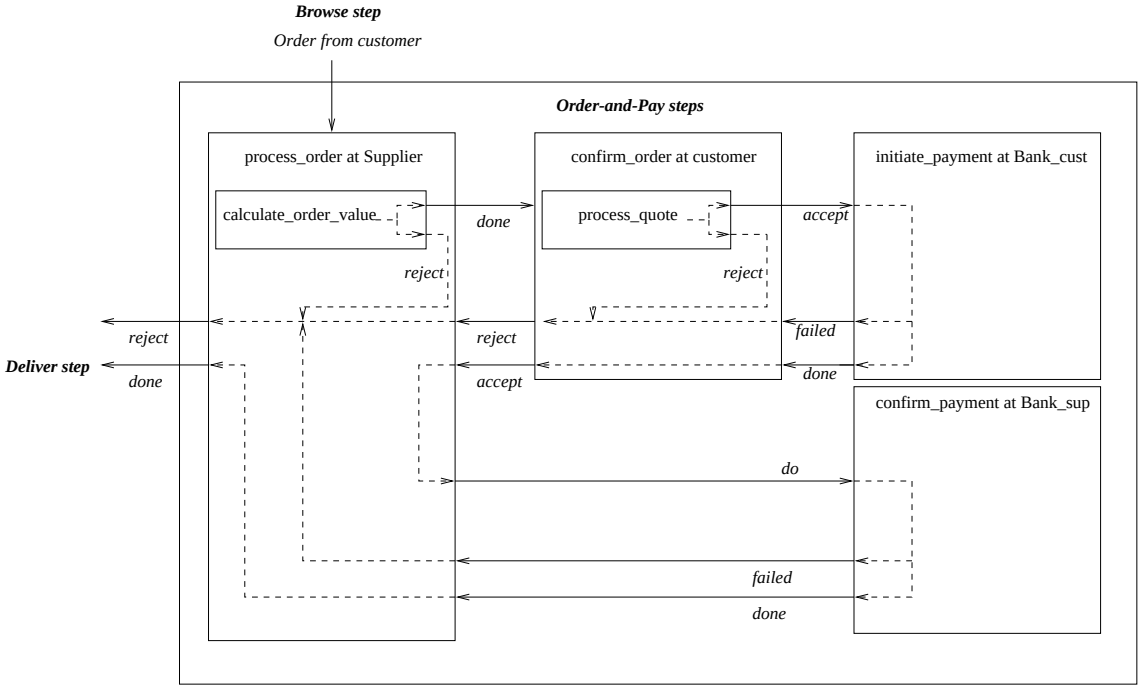
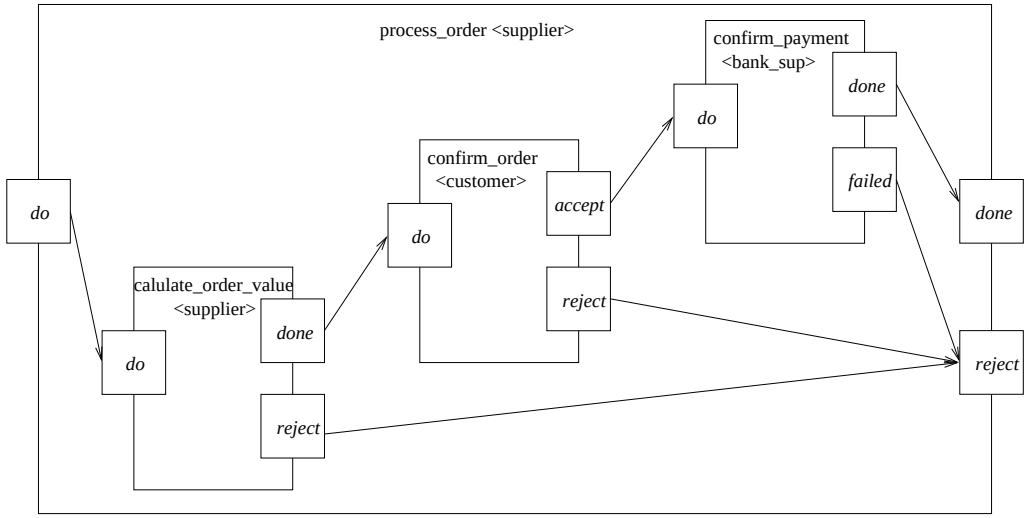
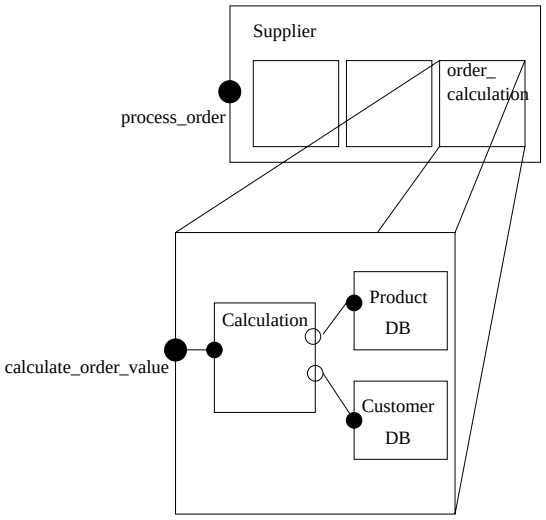



Figure 3: The abstract service schema for *order-and-pay* processing

As previously mentioned, service schemas actually correspond to workflow schemas and hence are described using the TCCS workflow language (see *Deliverables C2.1*, *B1.2* and *B1.3*). Figure 4.(a) gives the resulting graphical representation for the *process_order* workflow schema [Shrivastava & Wheater 1999]. Software architecture description is given using the TCCS ADL, which aggregates the ASTER, DARWIN, and OLAN ADLs where the two first provide notations for performing architecture analyses, while the third offers notations for base structural definitions and configuration deployment (see *Deliverables B1.1*, *B1.2* and *B1.3*). For illustration, Figure 4.(b) gives a graphical representation of the software architecture relating to the supplier organization. In the figure, boxes represent component and circles represent ports, black (resp. white) ports corresponding to provided (resp. required) services. The supplier embeds at least two services as required by the workflow schema: *process_order* and *calculate_order_value*. The former translates into a workflow schema while the latter is a primitive workflow task that interfaces with a base service of the supplier.




(a) The *process_order* workflow schema




(b) The *supplier* software architecture

Figure 4: Sample of workflow schema and software architecture descriptions for *order-and-pay* processing

This service is a complex component, i.e. a configuration (or software architecture) that is made out of a set of interacting components. Let us notice that the proposed graphical representation does not distinguish between ports corresponding to workflow tasks and those corresponding to operations provided by components. Such a distinction is however explicit from the overall description of a given complex service, which comprises the service's workflow schema (service's behavioral view) and associated software architecture (service's structural view).

## 3.2 Analyzing the service's structural and behavioral views

In order to help designers assessing design decisions, the TCCS environment provides CASE tools for service analyses. As pointed out in the bibliography from the software architecture domain (see *Deliverable A3.1*), a major advantage of complex software development based on software architecture description is that it enables thorough software design and analyses, based on formal methods. In general, methods that have been proposed in the literature further come along with CASE tools that non solely aid the designers' task but also make accessible those methods to practising engineers that are not experts with formal techniques. In the context of the C3DS project, we focus more specifically on tool support for behavior and QoS analyses due to the expertise of the project partners in this area. In addition, we are interested in applying provided analyses support to both structural and behavioral views of complex services. This constitutes one of the major contributions of the TCCS environment compared to existing ADL-based development environments. While those environments together provide extensive support for complex software design and analysis from the standpoint of the software's structural view, they offer limited support with respect to design and analysis of the software's behavioral view. In general, the latter aspect is tackled using model checking techniques (e.g. see the LTSA toolset provided by the TCCS), which enable specifying the behavior of interacting components. However, these do not address temporal dependencies among comments, which constitute a prominent aspect of complex service provisioning. This latter aspect is addressed in the TCCS environment through workflow schemas that may be conveniently analyzed with respect to liveness, safety and QoS properties.

**Behavior analyses:** Means for analyzing liveness and safety properties are provided in the TCCS environment through the LTSA toolset. Use of the LTSA toolset for the behavior analysis of software architectures has formerly been presented in *Deliverable B1.1.* Basically, such an analysis consists of defining primitive components in terms of finite state processes using action prefix and choice. The behavior of a composite component that is constructed from interconnected instances is then the parallel composition of the processes defining the instances (i.e. the process given for the component corresponding to the instance). Given the above behavior specification of components, the LTSA toolset allows the user to explore different execution scenarios by displaying the trace of actions resulting from a given set

of actions. The toolset further enables checking general properties concerning a software architecture, with respect to safety and liveness properties.

As detailed in *Deliverable B1.3*, the LTSA toolset may further be exploited for checking workflow schemas against safety properties. The undertaken approach to modeling workflow behavior is similar to that of architecture behavior. Basically, the interface set of a workflow task is modeled as the parallel composition of processes, which correspond to the task's interfaces, and inbound and outbound notifications. A specific primitive task is then defined as the parallel composition of instances of its input and output interface sets plus a default implementation process that prescribes the primitive behavior of any task. Finally, the behavior of a workflow schema is defined through the parallel composition of the embedded task instances. Given the above specification of a workflow schema, the LTSA toolset may be exploited for simulating its behavior and for checking safety properties.

**QoS analyses:** QoS analyses in the TCCS environment are supported through the ASTER toolset. As discussed in *Deliverable B1.1*, QoS analysis is in particular aimed at easing the development of middleware customized for the applications based on the required non-functional properties. This issue relates to the implementation of complex services, and is thus further addressed in the next section.

In addition to QoS analysis aimed at middleware customization, tool support for the QoS analysis of complex services regarding temporal properties is being investigated for both structural and behavioral views, within the TCCS environment. Temporal analysis of software architectures is detailed in *Deliverable B1.3*, concentrating more specifically on architectures relating to multimedia applications. In this framework, architectural elements handle data streams and binding among them set data-flow dependencies. Temporal analysis then relies on specifying temporal properties associated to components (i.e., timing guarantees between two successive output frames, and between input and output ports within a components; temporal constraints on time distribution between arrivals of frames), and on the composition of these temporal behaviors. The architectural style underlying the proposed temporal analysis is actually close to a workflow schema from the standpoint of describing data-flow dependencies. Hence, extension of the proposed approach for temporal analysis of workflow schemas is being investigated, as further discussed in *Deliverable B1.3*.

**Example:** Considering our example for order-and-pay processing, the above analysis toolset leads to annotate software architecture and workflow schema descriptions with corresponding property specifications. These specifications are then provided as input to the TCCS analysis tools (i.e. LTSA and Temporal analysis tools). This analysis sub-phase is depicted in Figure 5, the reader being referred to the aforementioned deliverables for details regarding corresponding annotations.

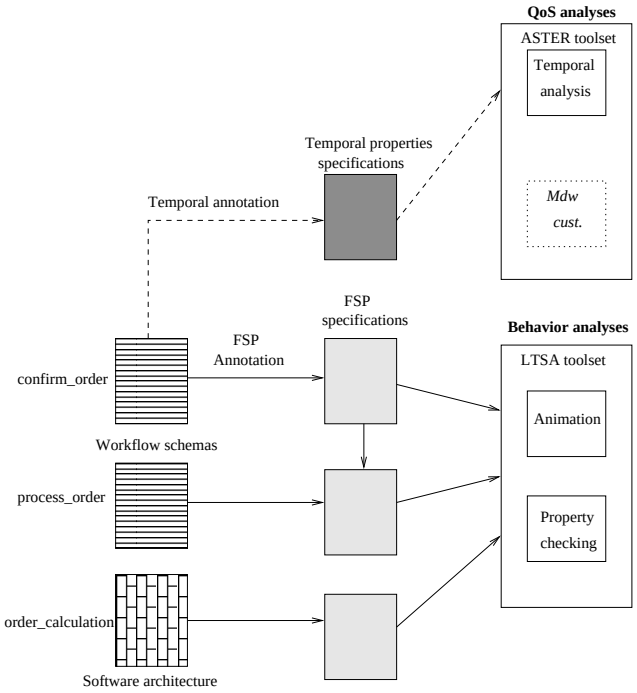



Figure 5: Analyzing complex services using the TCCS environment

# 4 The Implementation Phase

Once the complex service design has been thoroughly assessed, the implementation phase is provided with the description of the corresponding structural and behavioral views. During this phase, the developer is in charge of implementing those views together with the underlying middleware (see Figure 6).

## 4.1 Implementing the service's structural and behavioral views

Implementation of the complex service's structural and behavioral views relates to exploiting the functionalities of the TCCS platform, i.e. using the overall features as offered by the base CORBA infrastructure enriched with software agent and workflow management systems (see *Deliverable C3.2*).

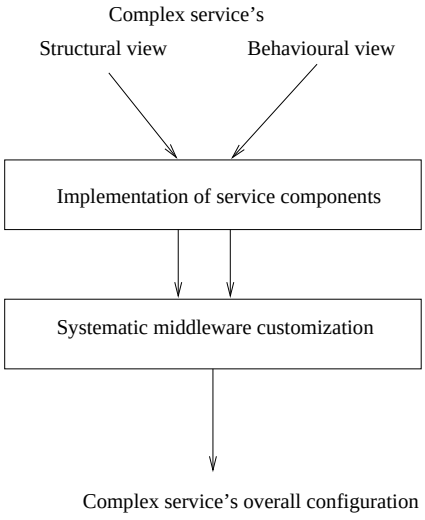



Figure 6: The C3DS implementation phase

**Structural views:** Implementing the service's structural view lies in providing implementation for the components that were added during the design phase, i.e. components that did not belong to the service repository. Such components may be either CORBA objects or software agents and will be developed according to the TCCS platform specifics.

**Behavioral views:** From the standpoint of implementing the service's behavioral view, this is quite straightforward using the TCCS workflow management system. The provided workflow schema is used directly by the system for execution. In particular, let us notice that the repository of workflow schemas that comes along with the system is the one used within the service repository (see Figure 2). The task left to the developer is then the implementation of the *task* objects, which wrap the service components implementing primitive tasks so as to interface them with the workflow management system.

**Example:** For illustration, Figure 7 depicts a possible implementation for the *order_calculation* component that was depicted in Figure 4.(b). In this implementation, we assume that the component is complex and that the *calculate_order_value* primitive task is realized through the implementation of four new components (i.e. 3 software agents and a CORBA object) combined with interactions with available database systems. This implementation further introduces wrapping components among the various management systems composing the TCCS platform, including those relating to the execution of workflow scripts and software

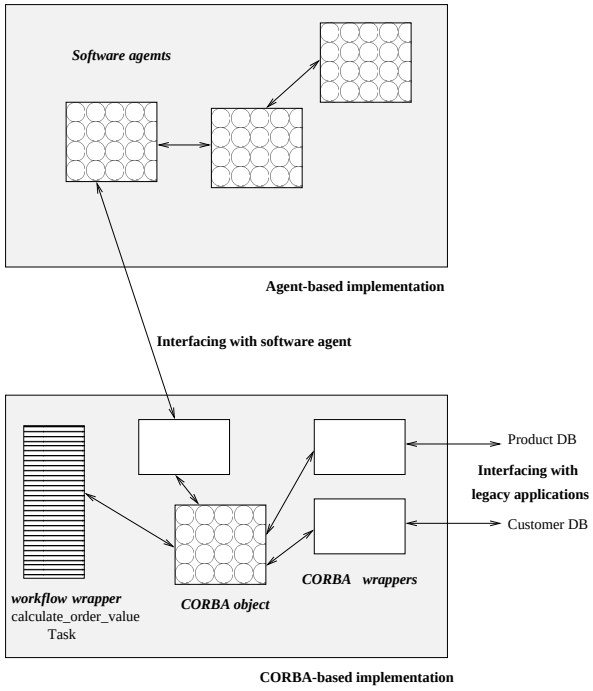



Figure 7: Implementing a complex service

agents.

## 4.2 Customizing middleware for complex services

As stated in the previous section, C3DS work on QoS analysis includes the provision of automated support for the customization of middleware according to the non-functional requirements of complex services. Latest results in this area are further detailed in *Deliverable B1.3*, and are outlined below. Let us point out here that as for the TCCS support for service design analysis, the one for middleware customization accounts for both structural and behavioral views of complex services, the latter being under study. Hence, this extends existing results in the area of software architecture to the domain of workflow.

**Customizing middleware for service components:** Base services used for complex service provisioning may be built through a number of interacting components (i.e. base services may be described as a software architecture). Interaction among components is then achieved using the communication services of the TCCS platform. However, various

non-functional properties may be enforced over the interactions using complementary middleware services, and in particular available implementations for CORBA Common Object Services (COSs). Support for middleware customization then consists of abstractly characterizing required non-functional properties for interactions within software architecture description. Such a specification is then used by the ASTER toolset for the systematic customization of the required middleware through the selection of adequate middleware services and generation of corresponding proxies for service components. In the framework of TCCS middleware, customization is addressed for the CORBA infrastructure, which offers a number of middleware services. On the other hand, middleware customization for software agents is not being investigated.

**Customizing middleware for workflow execution:** Just like the TCCS middleware may be customized for base services, the middleware used for the execution of workflow schemas may be customized so as to enforce various non-functional properties in addition to the *acidity* of transaction management. More specifically, the execution of a workflow schema lies in interactions among the *task controllers* associated with the workflow tasks. Currently, those interactions rely on the TCCS-based middleware comprising the TCCS ORB and OTS COS. However, additional non-functional properties may be required for the interactions. In particular, let us mention middleware customization with respect to security due to interactions among distinct organizations.

**Example:** Figure 8 depicts the process of middleware customization for the *process_order* workflow schema and *order_calculation* software architecture. Basically, the workflow schema and software architecture descriptions are annotated with non-functional requirements upon interactions, which are then provided to the ASTER toolset for middleware customization. The figure further gives an overview of the resulting customization process for the workflow schema; interactions among the workflow objects take place over the middleware comprising the TCCS ORB, CCS COS, and authentication service.

# 5 The Deployment Phase

The result of the implementation phase is the overall configuration of the complex service. This configuration embeds the software components relating to base services, workflow execution, and middleware services. The C3DS deployment phase then lies in instantiating these components over participating sites, together with the integration of additional services for supporting complex service monitoring and evolution (see Figure 9). Let us notice here that the C3DS deployment phase is currently under investigation and is subject to further elaboration.

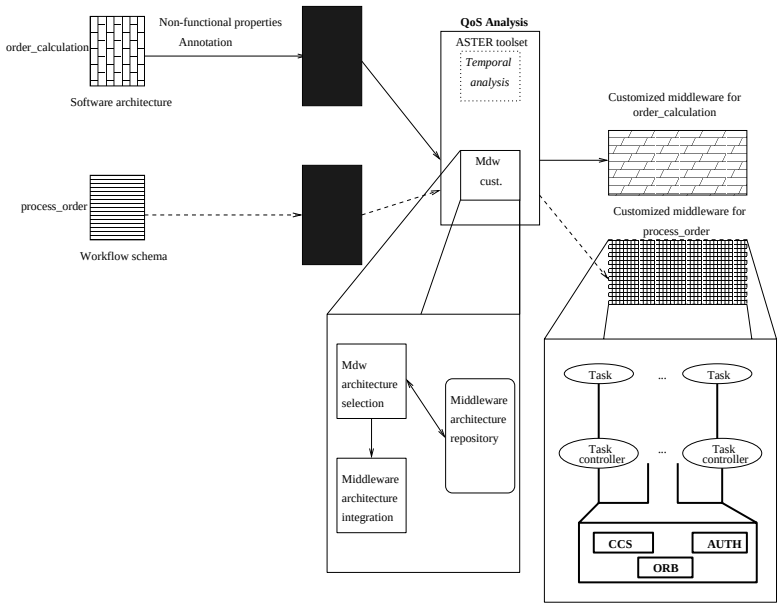



Figure 8: Customizing middleware for complex services

## 5.1 Initial deployment

The initial deployment phase consists of setting the distribution pattern for the complex service's configuration given the underlying distributed system (characterized as *Distributed system view* in Figure 9), i.e., the various organization sites from which available base services originate. Currently, placement decisions are left under the responsibility of developers and administrators, as discussed below. Providing tool support for systematic configuration distribution is an area for future research.

Deployment of the software components relating to the implementation of base services is achieved using the OLAN support, which enables annotating software components with location information (see *Deliverable B1.3*). Components relating to workflow execution subdivide into task and task controller objects. The former are associated with base services, and hence each task object is co-located with its corresponding base service. With respect to the latter objects, a novel feature of the Workflow system is that task controllers of an application can be grouped in an arbitrary manner. The coordination scheme could be distributed, where a task controller is co-located with its corresponding task object, or a centralized scheme, where the controllers have been grouped together at a given machine. A suitable configuration can be selected using the workflow administration application that is responsible for instantiating a schema. The choice of a given schema could depend on various

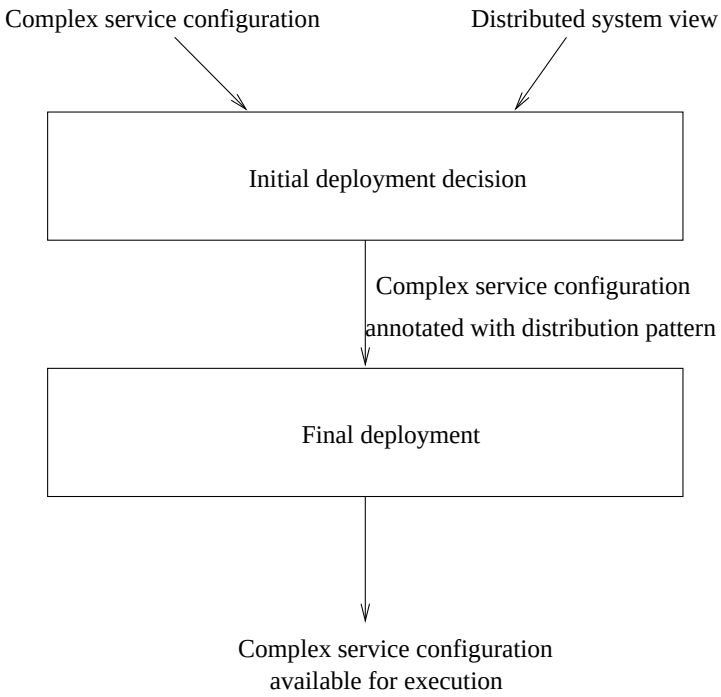



Figure 9: The deployment phase

factors (e.g., dependability, performance, monitoring, administrative convenience etc.), and is left to the users and administrators. If dependability is crucial to the workflow application, then the task controllers can be placed on multiple machines so that the failure of a single machine will have a minimal effect on the progress on the workflow application. If the monitoring of the progress of the workflow application is more important than its dependability, then the task controllers can be grouped on the monitoring machine so reducing communications overhead. In most cases the placement policy for the task controllers within the workflow application will be a compromise between these two extremes.

## 5.2 Final deployment for service evolution

The initial configuration deployment as discussed below does not account for complex service evolution due to change of either user requirements or environmental factors. The latter aspect is partly addressed during workflow execution due to the fact that the corresponding schema may embed exception handling and that workflow executions are transactional. However, further support is to be investigated so as to support the evolution of base services

(i.e. dynamic reconfiguration) or even of workflow schemas (e.g. revised definition of the schema). The C3DS project is addressing the above issues by investigating tool support for service monitoring and dynamic reconfiguration.

# 6 Conclusion

This report has presented the C3DS design and development methodology that lies in exploiting the provided TCCS environment for the provisioning of complex services out of existing ones. In that context, complex service design relies on the complementary description of the service's structural view and dynamic view. The former prescribes the service's software architecture, which abstractly characterizes all the base services composing the complex service. The latter defines the workflow schema to be applied onto the software architecture so as to actually provision the complex service. While there is an extensive body of research work on analysis support for software architecture design, analysis support for workflow schemas has been barely examined. The TCCS environment provides support for both analyses, applying results from the software architecture domain to the workflow domain. Implementation of a complex service from its design then lies in implementing the missing software components. In addition, the TCCS environment provides tools for systematic middleware customization with respect to non-functional requirements over interactions among the base services' components but also among the software components managing the execution of workflow schemas. The result of the complex service implementation is the service's overall configuration, which is to be deployed among the participating sites. The TCCS environment as discussed in this report is currently under elaboration. The current prototype includes a significant part of the environment constituents. Further development is still needed regarding the application of the QoS analysis toolset to workflow schemas, and support for complex service evolution. In addition, work is under way for making the environment user-friendly by enabling environment usage through graphical interfaces.

# References

[Baresi *et al.* 1999] L. Baresi, F. Casati, S. Castano, M.G. Fugini, I. Mirbel and B. Pernici. WIDE Workflow Development Methodology. In Proc. of WACC'99 – Joint Conference on Work Activities, Coordination and Collaboration. 1999.

[Cardelli 1998] L. Cardelli. Abstractions for Mobile Computation. In Secure Internet Programming: Security Issues for Distribution of Mobile Objects. 1998.

[Shrivastava & Wheater 1999] S. K. Shrivastava and S. M. Wheater. OPENflow Demo: E-

Commerce (Internal C3DS working report). 1999.

[Weske *et al.* 1999] M. Weske, T. Goesmann, R. Holten and R. Striemer. A reference Model for Workflow Application Development Processes. In Proc. of WACC'99 – Joint Conference on Work Activities, Coordination and Collaboration. 1999.