Workflow Modeling and Execution with Coloured Petri Nets in COSM

M. Merz, D. Moldt, K. Müller, W. Lamersdorf

University of Hamburg
Department of Computer Science
Vogt-Kölln-Str. 30 - D-22527 Hamburg, Germany

[merz|moldt|kmueller|lamersd]@informatik.uni-hamburg.de

Abstract

Modern distributed organisations use data communication increasingly fast, globally and at decreasing costs. Such developments facilitate flexible access to decentralized services in open networks and also allow coordination of complex interorganizational tasks as distributed applications.

This paper describes concepts and recent extensions to the distributed systems architecture COSM (Common Open Service Market) which supports integrated design, implementation, and execution of both access to arbitrary services in open networks and concurrent workflow modeling.

Formal basis of the workflow model are Coloured Petri Nets (CPN) which provide powerful means to specify and verify activity coordination in concurrent environments. CPN specifications, however, usually lack automated tool support for transforming them into efficiently executable applications. COSM, on the other hand, already provides a generic platform to manage distributed services (i.e. their properties, interfaces, etc.) in order to enable human users to engage easily in ad-hoc sessions with arbitrary service providers in open network environments. Extending COSM service descriptions with CPN workflow representations integrates dynamic workflow modeling into the system supported COSM service management platform. It could therefore help to bridge the gap between concrete system support for open distributed applications and formal models of concurrent workflow specifications.

This paper presents both a brief review of COSM concepts and prototype implementation details of its first CPN extensions, motivated by a workflow example from a real-world project environment.

Keywords: *Distributed Systems, Client/Server Computing, Workflow Management, Concurrent System Specification, Coloured Petri Nets*

1. Introduction

Todays global communication infrastructure allows to interconnect intercontinental organizational networks at fairly moderate communication costs. Internet information service infrastructures such as the World Wide Web (WWW), owe their success to both stable and ubiquitous availability of communication end-points and trivial infrastructure communication protocols. Such infrastructures are *open* not only in a technical sense - they are *organizationally open* since no application- (and organization-) specific adaption is required to access arbitrary distributed services with generic tools from anywhere in the network. On the other hand, so-called *configured* distributed service applications expect typically a specific communication behaviour from their users. Such semantic coherence is required for statefull servers as, for example, assumed in the *Remote Database Access* (RDA) protocol specification. In the long run, however, an increasingly large number of application-specific protocols can be expected which all require their users to obey application-specific regulations as, e.g., operation call sequencing. (An example for such applications was already presented in the "car-rental-service" of [1].)

Important general benefits of simple WWW-like applications are low service setup and access costs. For such applications, human users as service clients are not required to implement or install service specific access tools. In economic terms, such information service infrastructures impose low *transaction costs* upon accessing users, and therefore reduce the overall costs of service access - thus encouraging users to access remote service providers. For implementing configured distributed applications with individual access protocols, however, such simple platforms reach their limits - i.e. general set-up costs for such applications are (too) high.

1.1 Common Open Service Markets (COSM)

In this paper, the COSM (Common Open Service Market) system supports access to dynamically emerging statefull network services with a *single* generic tool (the *Generic Client)*. The result is a reduction of access costs even for network-services with specific, application-dependent protocols. In COSM, all interface (incl. protocol) aspects of remote services are formally described and transferred as a 'first-class' object to the Generic Client which in turn interprets these specifications and allows human users to interact with the remote servers conforming to their respective service interface and behavior specifications. The interaction of an individual Generic Client with such remote servers is modeled as a finite automaton or finite state machine (FSM). For distributed workflow execution the specification of concurrent activities should also be possible. In order to model concurrent workflows adequately, first a formal modeling technique is required. In this context *Petri Nets* (here: Coloured Petri Nets) are used.

Based on the possibility to specify, for example, 'legal' invocation sequences of remote services, the following sections focus on the extension of COSM specification mechanisms to support a transition from workflow *modeling* to an immediate *execution*. The specific contribution of the COSM architecture is the ability to set up a distributed task at low configuration costs. Therefore, even extra-organizational workgroup members can be involved at low setup effort as required by the following example.

Beyond the problems addressed in this contribution, the COSM project focusses on further research fields: Embedding trusted third party services into the infrastructure, like notary services, which assure a secure and confidential legal contraction support for client and service components on an electronic market [2]. Within a co-project of COSM, *TRADE* (TRAding and coorDination Environment), service access is mediated by a distinct service mediation mechanism - the *trader* [3]. Components of both environments will share mechanisms described in this contribution in order to support activity coordination.

1.2 A simple workflow example

In order to clarify the application background of the following sections and to present a case study for our approach, we refer to a realistic workflow example from an airline application domain: an airline is obliged to create a 'Flight Report' (FR) for each flight carried out. This document reports "unusual events" within the technical domain or among passengers. It is created by crew members after flight and posted to a central *FR-Pool*. FRs are examined periodically, evaluated by a *FR manager,* and assigned to one or more experts, depending on the kind of occurrence. These experts are usually from the technical domain, the security domain, or financial accounting staff members and may be located at remote branches or even different companies as, e.g., a parts manufacturer. The experts involved make a statement on the subject and post results back to the FR manager. Statements are made independently from each other, thus allowing concurrency. After all statements have been delivered, the FR manager classifies the FR and decides how to proceed. Two possibilities exist: Either the FR contains no problematic aspects or an urgent meeting to tackle the problem is neccessary. In the first case the FR is posted to be filed on optical media for legal reasons. In the latter the meeting is initiated and the FR is returned to the FR manager in order to obtain additional statements.

1.3 Organization of the Paper

The following section of this paper briefly reviews concepts for distributed application execution in COSM. Section 3 introduces Coloured Petri Nets as a modeling and simulation technique for concurrent activity management and control. Then, section 4 outlines the integration of a Coloured Petri Net concept and its exploitation by COSM components. Section 5 finally describes some details of an ongoing COSM prototype implementation.

2 Application execution within COSM

A general problem of distributed application design is to find a suitable distribution level between the top-most presentation level and a remote file access as the most bottom solution of an application. Both extreme solutions imply high network and process load. Therefore, the COSM approach is to shift the function split of application code between the client and server part of a distributed application to an optimal level that reduces load, on one hand, and allows the Generic Client component to remain service-independent, on the other. Accordingly, to interact with a remote server, the user binds to it by receiving a COSM *service representation* (SR) at run-time. A SR is both a capability to access that server and a description of the service interface and semantics (See Figure 1).

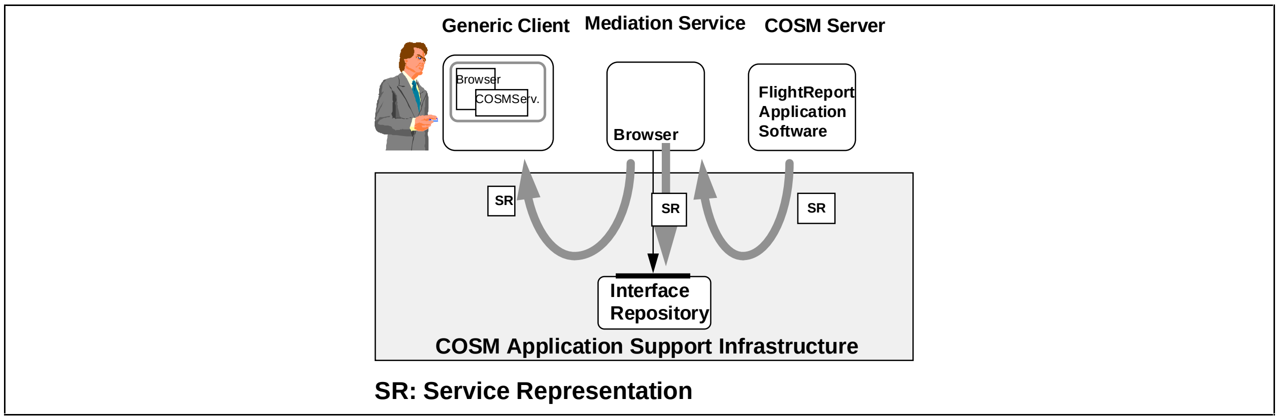

Generic Client
Mediation Service
COSM Server
Browser
COSMServ.
Browser
FlightReport Application Software
SR
SR
SR
Interface Repository
COSM Application Support Infrastructure
SR: Service Representation

Figure 1: Service access in COSM

2.1 Access mechanisms to open services

The COSM *Generic Client* (GC) is a tool for human users to acquire remote server access and to support interactions between users and servers. Communication between client and server is based on a Dynamic Invocation Interface (DII) as defined within the CORBA (Common Object Request Broker Architecture) specification of the Object Management Group [4]. The DII dynamic typing mechanism immediately results from ad-hoc binding requirements at the COSM-application level. Although late binding generally implies the possibility of a type mismatch at run-time, communication in COSM is type-safe since RPC invocations are generated from SR operation signature descriptions. RPC data types correspond with user interface editor types in order to allow users to inspect and modify parameter and result values. Besides standard DII data types, COSM provides special types for, e.g., service references, opaque values, and ASCII files. By supporting service references as first-class data objects COSM servers are able to display directories of servers via the Generic Client to the user - similar to WWW hypertext references. Binding to a such referred server is effected directly from the user interface by selecting the according service reference. Loosely coupled chains of servers may emerge from this principle (Figure 1).

2.2 Service Representation

At its most generic level, the SR is a container for data structures of arbitrary types at run-time. An SR-interpreter (as a component of the Generic Client) accepts a set of components describing the service such as:

- A specification of *operation descriptions*, containing operation names and *parameter descriptors*. Parameter descriptors refer to *data objects* which contain actual values.
- A specification of the *user interface* to be generated for human users by the GC. It contains specifications for dialog boxes, data editors and push buttons. Data editors manipulate data objects.
- A specification of the *service interface protocol*, i.e. which operations are enabled to be invoked at a given state. Currently, this protocol description is based on a FSM model and comprises a set of application states and transitions between them which refer to operation descriptions. State changes are effected by user-level events and execute, in turn, RPC invocations.
- Informal description components,e.g., help texts to support human users.
- Cost information components that inform users on service access and operation invocation fees.
- Any data values locally used by the GC.

Since the Generic Client software component is application-unspecific, state information like, e.g., window positions or counter variables are captured by the service representation. Therefore it is possible to store the SR persistently and to suspend interactions with the remote server. This SR can be re-activated later - or from another network site - in order to resume the previous communication state.

Two benefits arise from an SR-based approach to implement open service access: first, *representations* are standardized within COSM, not interfaces. This includes a uniform, service-independent user interface presentation and allows type-safety data entry. Second, more application-specific functionality can be shifted to the client site. In fact, COSM servers are reduced to plain call libraries with corresponding SRs. Compared with, e.g., X-Windows, dialog control is shifted to the user part of a distributed application.

The Generic Client dialog management component connects presentation functions to the application call interface. Users are allowed to invoke remote operations by pressing corresponding buttons. In the case of state-prone services buttons are disabled if the operation is not allowed to be invoked in the current application state.

As containers for arbitrary data types and values, SRs are extensible at run-time. This allows specialized environments - as will be presented later in the case of workflow modeling - to introduce dedicated component types for their own purposes. COSM applications that are not aware of these extensions will still be allowed to interprete the subset of SR components they are specialized on.

3 Workflow modeling with Coloured Petri Nets

Before introducing Coloured Petri Nets as a specification method a definition of the term *workflow* is given.

In our model a workflow consists of a set of activities A, a set of roles R and a mapping $E : A \rightarrow R$ which assigns an *enabling role* to each activity. Human users may act within several roles, therefore, we define a N:M relation between users and roles. Finally, a workflow description contains the *control flow* between activities. This defines a precedence relation between adjacent activities. An activity is enabled, if all preceeding activities are carried out. Each single activity A comprises the following components:

- An *action,* as the actual core activity. Actions are realized as operation invocations at remote services in COSM.
- A *precondition:* If this predicate evaluates to *True* the activity can be carried out.
- *Input data,* required for the execution of actions.
- *Output data* is the outcome of the action.
- *Postconditions* are set as triggers for succeeding activities.

Activities that take place concurrently may share resources if they do not conflict. Further, besides the control flow between single activities a workflow description implies communications between them - the *data flow*. In our model, communication takes place indirectly via operations on data structures.

3.1 Coordination modeling using Coloured Petri Nets

Petri nets [5] serve as a graphical representation for workflow models [6]. Their advantage lies in the combination of a mathematical foundation, a comprehensive graphical representation, and the possibility to carry out simulations and verifications. In principle, Petri net tools allow to assure liveness and to proof deadlock absence for certain net classes. Formally, Petri nets are directed, bipartite graphs, in the simplest case consisting of a set of *places* P and a set of *transitions* T, and a flow relation (*edges*) F with F (S 5 T) ∪ (T 5 S).

To model system behaviour, places can be marked with tokens. In the simple case of *condition/event nets* tokens are *anonymous* and each place contains at most one token. A transition is *enabled* if all *input places* and no *output places* are marked with a token. To *fire* a transition it is enabled, tokens are withdrawn from all input places and all output places are filled. In the case of *Coloured Petri Nets* (CPN), however, tokens are typed (coloured) individuums or data values. Places, transitions, edges etc. are typed using an according signature. Places represent data stores which contain an arbitrary number of data values (tokens) of their respective type. Transitions are enabled in CPNs when

1. all input places provide tokens of the type that is associated with the respective input edge and furthermore all expressions, variables, and values evaluate conflict-free for the binding and

2. the transition predicate evaluates to "true". Complete definitions can be found in [5].

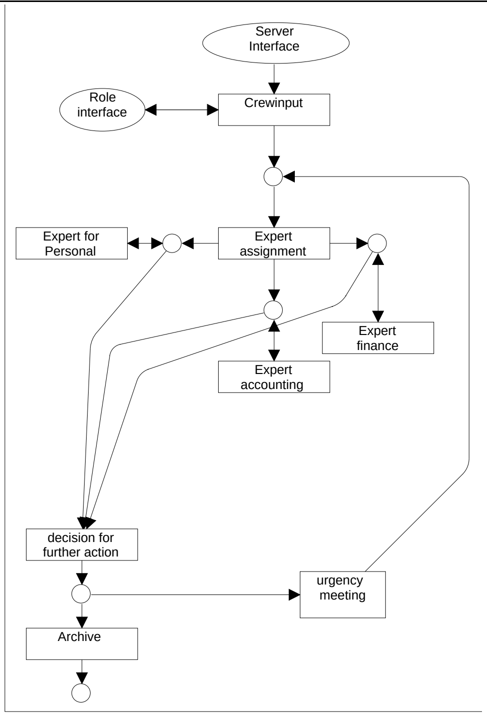


Figure 2: CPN Graph for the Flight Report Procedure

Figure 2 shows a graphical representation of a Petri net model for the workflow example introduced above.[1] The relationship between our workflow model and a corresponding CPN-based specification is as follows: States serve as data stores between activities and transitions map to individual activities. The precedence relation can be mapped to the flow relation and the mapping of the other constructs are described in subsection 4.1.2. E.g., role[2] identifiers are represented by places and their marking associated with transitions by the flow relation. In Figure 2 there is an example: transition `Crewinput` has a place `role interface` as a side condition. This place has to be marked for the enabling of the transition. The marking is put on the place by the GC, when the SR is received from the server. In the next section we explain the handling of the control flow within the GC.

3.2. Control flow for Generic Clients

In this paper we concentrate on the service representation and the COSM implementation. An overview of the environment, the SR is handled within, is presented in this subsection.

The GC organizes the treatment of the service representation in a generic way. For this an abstraction mechanism is used which is known as *Task/Flow-Systems (TFS)* [7]. This technique is based on condition/event-nets, where tokens are themselves condition/event-nets, marked, in turn, with anonymous tokens. TFS transitions represent *functional units* of systems. This allows a separation of executing units (here: GCs) and the tasks (here: SRs). In [8] it is shown in detail how to apply this approach to COSM. [9] describes the usage of this abstraction mechanism at a hierarchy of more than two levels. In [10] abstraction mechanisms for nets are developed for

[1] In our net examples we restricted the inscriptions to the minimum. The reader has to deduce the missing ones.

[2] In this model, user roles are not considered as a net extension since login procedures are a "general matter" and do not change with workflow models

CIM systems, where a similar problem was faced. It is important to notice that the involved people are not obliged to think in nets, even though this would be preferable and is intended here.

The control flow model for the GC is illustrated in Figure 3. From the AVN point of view, the assumption of this paper is that the server sends a Service Representation in form of a net to the Generic Client. As a functional unit, the GC adds some information which are relevant for the SR and provided locally. The extended SR is then used for the execution. The format of the SR is explained in subsection 4.1.2.

An example can be found in Figure 3: The incoming SR at place `server interface` and the added role information from place `logged in.`

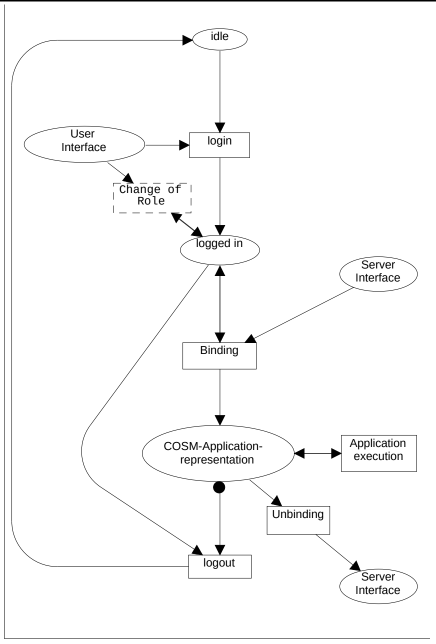


Figure 3: Contol Flow Model for Generic Clients

4. Integrating CPN representations into COSM

Principal tasks to integrate CPN modeling and the COSM infrastructure are presented in this section. The first concerns the representation level of COSM - it determines the transformation of CPN specification elements into SR counterparts. The second task is to enhance Generic Client and binding mechanisms to satisfy workflow control requirements. Finally, an application-level extension is required by introducing an additional *task server*: already existing reference and binding mechanisms of COSM are exploited to coordinate activities of participating clients.

4.1. Transforming CPNs into Service Representations

For the discussion of the representational level of COSM the existing SR schema has to be extended. On the other hand restrictions of the nets used are neccessary. After explaining the restriction the mapping from CPN to COSM is presented.

4.1.1. Adjusting the current SR schema

The current COSM design supports the specification of an interaction protocol for single user sessions: in such case there are no means required to specify concurrency. The current Generic

Client design is thus based on a FSM model. To capture concurrent activities in the workflow model, additional components have to be added to the SR and the SR interpreter has to be enhanced accordingly. "Traditional" automaton-based GCs are still capable to interprete the automaton-based protocol specification whilst only GCs specialized as workflow participants are aware of CPN-extended SRs. SRs can thus be considered as data object with a polymorphic type system where a given SR containing automaton *and* CPN definitions is a subtype of each of the requested types "automaton-based SR" and "CPN-based SR".

4.1.2. Net restrictions for COSM

In this paper there are also some restrictions for the Petri net based approach. The concept of concurrency is restricted to fit into the easily transformable model of FSMs. This is neccessary to realize an approach which allows a smooth transition from specification to implementation. The basic idea is as follows: Modelers build a sequential FSM model, transform this into a net, check it semantically for possible concurrency, and modify the net accordingly.[3]

General net restrictions are: The place capacity within the SR is one token. This token can still be a complex data structure. The structure of the SR net is mainly following the FSM. Concurrency is introduced by splitting specific conflict places according to the involved independent transition. As an example see the output places of transition `expert assignment` in Figure 2.

To transform CPN representations of design tools into SRs the following mappings are required:

	CPN representation component	*SR component*
1)	CPN token types for places	SR type components
2)	Presence of tokens at places	*Enabling flag* of type *Boolean* which indicates the presence of a token in the place
3)	CPN token values within places	Usual SR data objects
4)	CPN transitions	SR transition components containing: Lists of enabling flags for input and output transitions, a predicate component, a role reference, and a reference to the corresponding operation description component
5)	CPN transition names	SR operation names
6)	CPN names of input and output edges	SR parameter and result names for each operation
7)	Referenced input and output states for each transition	References to SR data components containing parameter and result values
8)	CPN initial marking	SR initial setting of enabling flags and data value initialization

Table 1: Transformation of CPN representation into SR

After transforming a CPN representation into SR components further information has to be appended. First, enabling roles are assigned to each transformation. A dedicated *role editor* is used for this step. Accordingly, additional description components will be inserted into the SR which refer to previously defined components, e.g., the GUI interface description, which specifies direct manipulation of SR data values and GUI elements to invoke remote server operations. Figure 4 shows the mapping of model-level entities to SR components, represented in a textual CORBA-IDL-like notation for reasons of comprehensiveness:

[3]The net model could of course be build directly with the same restrictions.

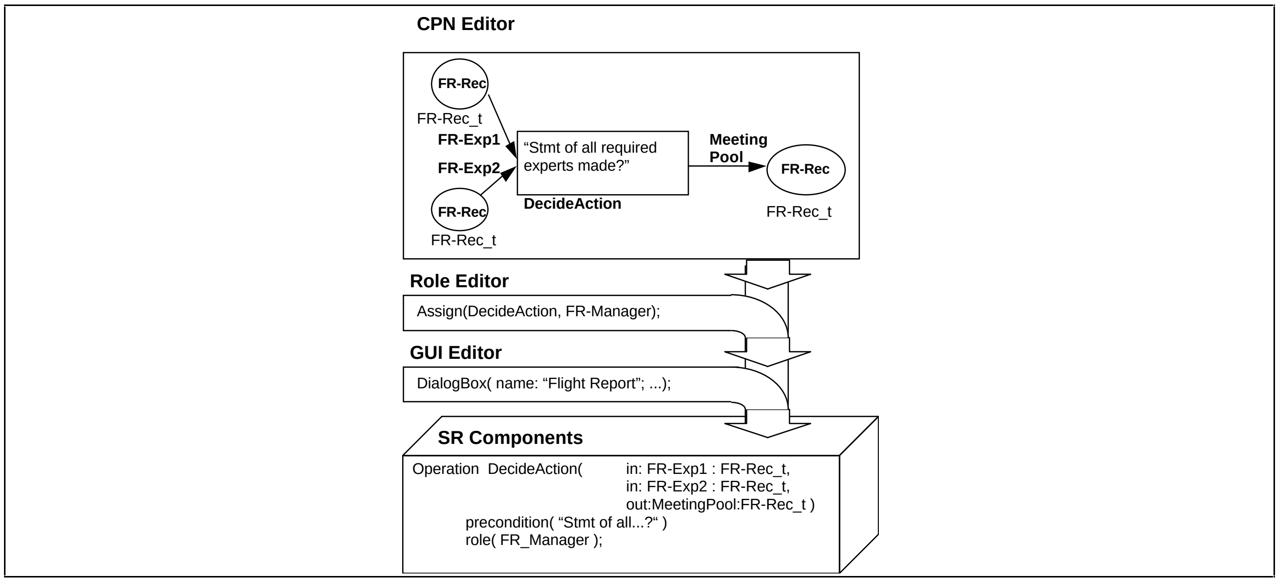

CPN Editor
FR-Rec
FR-Rec_t
FR-Exp1
FR-Exp2
"Stmt of all required experts made?"
DecideAction
Meeting Pool
FR-Rec
FR-Rec_t
FR-Rec
FR-Rec_t
Role Editor
Assign(DecideAction, FR-Manager);
GUI Editor
DialogBox(name: "Flight Report"; ...);
SR Components
Operation DecideAction(in: FR-Exp1 : FR-Rec_t, in: FR-Exp2 : FR-Rec_t, out:MeetingPool:FR-Rec_t)
precondition("Stmt of all...?")
role(FR_Manager);

Figure 4: Deriving service interface specification from the CPN net representation

4.2. Service infrastructure extensions for COSM workflow support

Additionally, the capability of users to execute invocations is further restricted by role annotations in the COSM workflow extension. To obey this information, Generic Clients have to be extended to provide a user authentication procedure. Only if role requirements are satisfied, transitions can be fired. After an activity has been carried out the user may - if possible - switch the role or deliver the SR back to the server.

To identify the current user a role identifier has to be appended to the COSM binding protocol data unit. Otherwise servers are not able to send role-specific task lists back to the Generic Client.

Since several tasks may be carried out by different users at the same time they are assigned with a unique task identifier which is represented as an additional SR component.

Concurrency is allowed at the model level if more than one transition is enabled and they do not conflict with one another, e.g., in the case of experts that make statements independently. To reduce complexity arising from such cases, one SR is associated with only one single thread of activity. Concurrency remains allowed to occur between several Generic Clients. However, this requires extensions of the server implementation at the application level. If a CPN transition inserts tokens into more than one output place, potential for concurrency is given. Now enabled transitions may fire concurrently at the model level. After the user has send back the SR to the server, the SR is replicated according to the number of enabled transitions so that further invocations based on these SRs may take place concurrently. When concurrent activities have to be synchronized the interpreter assures correct behaviour by firing transitions.

4.3. Application-level extensions: The Task Server

Several experts make their statement in parallel - each within an individual session at the Generic Client and therefore based on an individual SR. In order to support this concurrency and to coordinate server access, a COSM-server has to trace the workflow state to deliver SRs to the respective clients. To keep server implementation as simple as they are in the base COSM system, this coordination task should be delegated to a dedicated *task server* which is itself an additional COSM server that administrates SRs for a set of other application-specific servers.

SRs are inserted into role-specific task lists at the task server. By binding to this server a user gains access to specific tasks which are displayed via the GC. By selecting a task (resp. an SR) the user aquires the right to execute the operation which is enabled at the SRs current state. After carrying out activities, SRs are sent back to the task server. See Figure 5.

In the case of concurrent activities, for example after the transition `ExpertAssignment` in Figure 2, task servers have to split an SR into several copies and to insert these into the according task lists. Correspondingly, these copies have to be merged after all concurrent activities have been carried out. To prevent any integrity conflicts, concurrent activities must have been specified conflict-free at the model level.

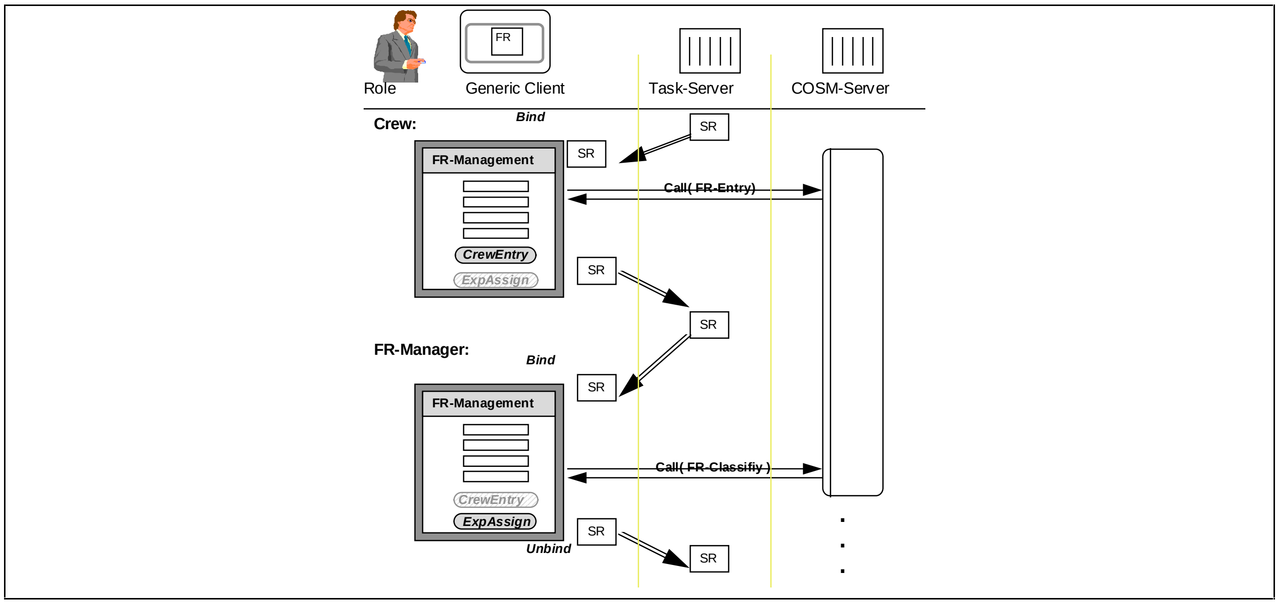


Figure 5: Control and Data flow at the application execution

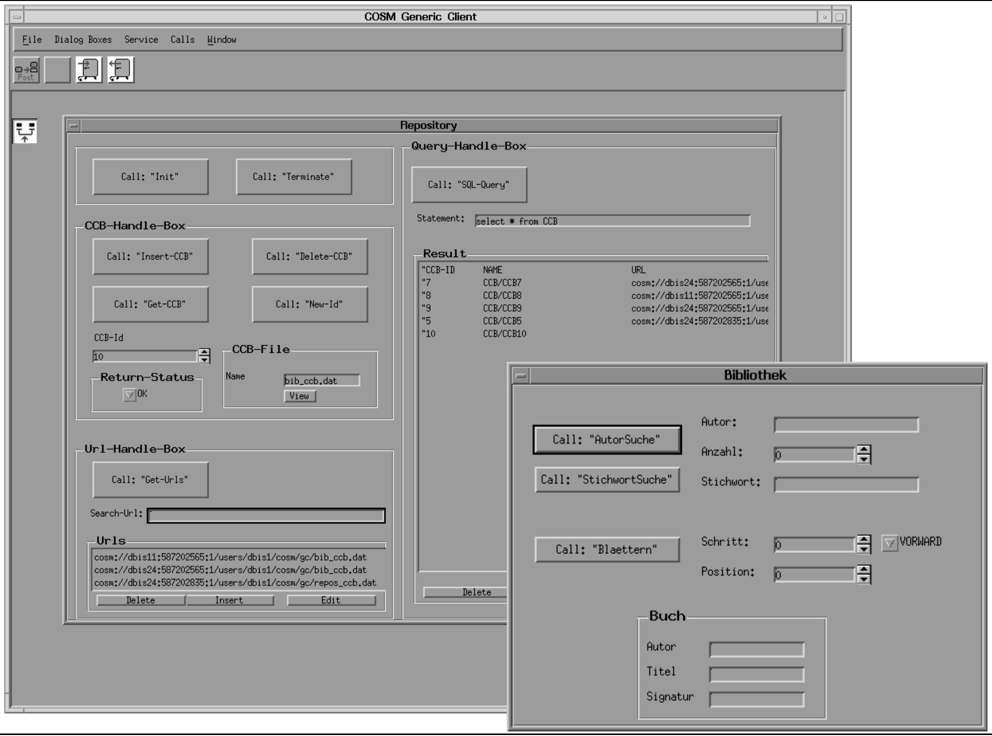
After logging in at the Generic Client, the user first binds to the task server. The user authentication is transferred to the task server to control the access to a role-specific task list. As in the case of the base COSM system (Section 2.2), the user is guided through the task server's operation invocations by the GC user interface. If a new workflow instance is to be obtained, an initialized SR with a unique task identifier is generated by the task server for the client. In the flight report example this takes place when the Crew is going to fill out an initialized flight report. In opposition to the base COSM architecture, the actual operation invocation is not carried out at the task server itself, but at the flight report server, which is identified within the SR. After having set up a new flight report, the task server passes control to the FR-manager after receiving the SR. As the next activity, the FR-manager examines the own task list, loads the SRs and carries out the `ExpertAssignment` operation.

The benefit of the direct transformation from workflow models to COSM service representations is to design the application specific workflow model, to prove features like liveness, deadlock absence, mutual exclusion in cases of conflicting transitions, and, as the last step, to automatically set-up control and data flow based on the workflow model.

5. Current prototype implementation

The current COSM prototype has been developed on a cluster of RS/6000 AIX workstations from IBM. It mainly consists of four component types:

- The *Dynamic Invocation Interface* was built upon the Sun RPC and XDR interface. It allows to allocate lists of structured parameter objects and to reconstruct them at the receivers site after transmission.
- The *Service Representation Manager* controls access to the binary SR data structure. The SR is organized as a contiguous memory allocation unit with local memory management.
- *Generic Clients* are built upon the previous components. They consist of a presentation layer, a dialog control unit and the *invocation manager*. In this order, a GC displays GUI information, handles user input events and invokes remote procedure calls based on information extracted by the service representation manager (See Figure 6).
- Several *server applications* for testing purposes. Several demo servers have been developed, for example an SR repository that allows servers to register their SR on the one hand, and Generic Client to access the repository database, on the other. Registered SRs are interpreted by the repository and stored among others in the database. Generic Client users, who are bound to the repository server may run queries against the database to obtain structured information on the SRs registered. (See Figure 6)



CCB-Id: 10

Return-Status: OK

Name: bib_ccb.dat

Statement: select * from CCB

```
"CCB-ID     NAME          URL
"7          CCB/CCB7      cosm://dbis24:587202565:1/use
"8          CCB/CCB8      cosm://dbis11:587202565:1/use
"9          CCB/CCB9      cosm://dbis24:587202565:1/use
"5          CCB/CCB5      cosm://dbis24:587202835:1/use
"10         CCB/CCB10
```

```
cosm://dbis11:587202565:1/users/dbis1/cosm/gc/bib_ccb.dat
cosm://dbis24:587202565:1/users/dbis1/cosm/gc/bib_ccb.dat
cosm://dbis24:587202835:1/users/dbis1/cosm/gc/repos_ccb.dat
```

Anzahl: 0

Schritt: 0

Position: 0

Figure 6: Sample Generic Client user interface

6 Conclusion and outlook

The COSM project aims at improved system support for flexible client/server integration in distributed and heterogeneous open systems. An important goal is to support not only *specific, predefined* client/server cooperations but rather to design a *generic* architecture for flexible service management, access and coordination in open systems. Especially, use of the CPN specifications for workflow modeling and a seamless transformation of CPN models into the COSM architecture and software infrastructure helps to address and solve different design and implementation problems in their respective distinct environments: at the *infrastructure* level, COSM addresses a flexible access and management of arbitrary remote services in general. CPN specification tools, on the other hand, are first used to model workflow control formally and then derive implementation support based on such specifications. Accordingly, this paper focusses on the integration of existing techniques from both areas in a common framework.

Current *practical work* is maily concerned with a continued prototype implementation of this integration in order to extend COSM to also support coordination of concurrent and distributed workflow patterns. Based on the workflow representation as described in section 4.1, the following additional COSM system components are developed currently:

- a *net interpreter* which allows to animate the service net representations.
- a *representation converter* which operates on programming interfaces of tools and transfers information about the defined net structure and marking into service representations.
- Tools like a *role and GUI editor* are planned to facilitate interactive generation of service representations.

In order to extend the available *formal* techniques, current work in this area concentrates on abstraction mechanisms like net-hierarchies and Task/Flow-Systems. The purpose here is to model encapsulated systems at an infinte level of layering. First approaches for such abstraction mechanisms for CIM systems are discussed in [10]. In addition, results from system theory and net theory have to be combined. An integrated formal mechanism could then be used for applications with a complex structure as, e.g., distributed systems, workflow support, CIM, systems analysis and concurrent system specification, which can be adequately described with high level Petri nets.

7. References

[1] M. Merz, K. Müller, W. Lamersdorf: "Service Trading and Mediation in Distributed Computing Systems", Proc. IEEE International Conference on Distributed Computing Systems, Los Alamitos 1994, pp. 450-457

[2] M. Merz, K. Müller, W. Lamersdorf: "Trusted Third Party Services in COSM", in "EM - Electronic Markets", No. 12, September '94, Volume 4, pp. 7-8

[3] M. Merz, W. Lamersdorf: "Cooperation Support for an Open Service Market", IFIP / GI 'International Conference on Open Distributed Processing' (ICODP'93), Berlin, August 1993, pp 329-340

[4] The Common Object Request Broker: Architecture and Specification, OMG Document No. 91.12.1, 1991

[5] Jensen, K.: "Coloured Petri Nets", Volume 1, Springer Verlag, Berlin Heidelberg New York, 1992

[6] Gruhn, V.: "FunSoft Netze", in: Scheschonk, G. and Reisig, W. (Ed.): "Petri-Netze im Einsatz für Entwurf und Entwicklung von Informationssystemen", Informatik Aktuell, Springer Verlag, Berlin Heidelberg New York, 1993, pp. 31-45

[7] Valk, R.: "Task-Flow Systems", Technical Report # 124, University of Hamburg, Computer Science, 1987

[8] Merz, M. and Moldt, D.: "Nebenläufige Modellierung von COSM mit Hilfe von Petrinetzen", in German, Technical Communication, University of Hamburg, Computer Science, to appear in March 1995

[9] Heinemeier, A. and Moldt, D.: "Nets in Nets - An abstraction mechanism for Petri nets", Technical Communication, University of Hamburg, Computer Science, planned for May 1995

[10] Borusan, A. and Moldt, D.: "Method for modelling CIM-Systems with Coloured Petri Nets", in: Cotsafis, M. and Vernadat, F., Ed., Advances in Factories of the Future, CIM and Robotics, pp. 91-100. Elsevier, Amsterdam London New York, 1993