A Workflow and Agent based Platform for Service Provisioning

S.K. Shrivastava[1], L. Bellissard[2], D. Féliot[2], M. Herrmann[2], N. de Palma[2], and S.M. Wheater[1]

[1]*Department of Computing Science, Newcastle University,*
Newcastle upon Tyne, NE1 7RU, England

[2]*SIRAC project and AAA-DYADE*
INRIA Rhône-Alpes, 655, Avenue de l'Europe
F-38330 MONTBONNOT SAINT-MARTIN, France

Abstract

The design and implementation of a dependable system that provides a composition and execution environment for distributed applications whose executions could span arbitrarily large durations is described. The objective is to create a framework for complex service provisioning. By complex service provisioning we primarily mean the ability to compose a given service out of existing ones as well as the ability to exercise dynamic control over the execution of the service. The approach taken is centred around building middleware services based on integration of workflow and agent technologies. The platform enables these two systems to interact via CORBA services. Service behaviour and service deployment are represented as workflow processes. Individual tasks that make up the workflow would be legacy applications, specially created tasks, and agent applications. Agents are able to create workflow instances, receive results from workflows and send inputs to workflows. This enables agents to act as user agents capable of managing workflows on behalf of users.

1. Introduction

We describe the design and implementation of a dependable system that provides a composition and execution environment for distributed applications whose executions could span arbitrarily large durations. Our objective is to create a framework for complex service provisioning. By complex service provisioning we primarily mean the ability to compose a given service out of existing ones as well as the ability to exercise dynamic control over the execution of the service. The types of services we have in mind are to do with the automation of so called 'business processes' of organisations that are increasingly using the Internet for their day to day functioning. The domain of electronic commerce is one well known example: here automation of business processes would include system support to help consumer-to-business as well as business-to-business interactions for buying and selling of goods. We take a broad view of service provisioning that includes the entire life cycle of a service: from its specification to deployment (creation) to management. By service management we mean managing the created service by online monitoring, control and dynamic reconfiguration and finally to its termination.

A number of factors need to be taken into account in the design of a framework for complex service provisioning. First, most applications (or services, we will use these terms interchangeably) are rarely built from scratch; rather they are constructed by composing them out of existing applications and protocols. It should therefore be possible to compose an application out of component applications in a uniform manner, irrespective of the languages in which the component applications have been written and the operating systems of the host platforms. Application composition however must take into account individual (site, organisation) autonomy and privacy requirements. Second, the resulting applications can be very complex in structure, containing many temporal and data-flow dependencies between their constituent applications. However, constituent applications must be scheduled to run respecting these dependencies, despite the possibility of intervening processor and network failures. Third, the execution of such an application may take a long time to complete, and may contain long periods of inactivity (minutes, hours, days, weeks etc.), often due to the constituent applications requiring user interactions. It should be possible therefore to reconfigure an application dynamically because, for example, machines may fail, services may be moved or withdrawn and user requirements may change. Fourth, facilities are required for examining the application's execution history (e.g., to be able to settle disputes). So, a durable 'audit trail' recording the interactions between component applications needs to be maintained.

Our approach to the creation of a framework for complex service provisioning is centred around building middleware services based on integration of two technologies: workflow and agent. We have built such a platform, called the Task Control and Coordination Service (TCCS) platform by making use of an existing dependable distributed workflow system (that uses CORBA middleware) and an existing dependable distributed agent system (that uses message oriented middleware, MOM). The TCCS platform enables these two systems to interact via CORBA services. Service behaviour and service deployment are represented as workflow processes. Individual tasks that make up the workflow would be legacy applications, specially created tasks, and agent applications. The platform provides interoperability at level of workflow tasks and agents. Agents are able to create workflow instances, receive results from workflows and send inputs to workflows. This enables agents to act as user agents capable of managing workflows on behalf of users.

The paper discusses how we provide flexible composition and dynamic reconfiguration facilities by making use of these two technologies. However, in seeking to obtain the best of both worlds by combining the two technologies, there is the real danger of confronting application designers with a bewildering choice of application development options – a situation that will almost certainly make the task of service provisioning harder rather than simpler. We discuss how we have avoided this pitfall by constructing a service development environment that provides a consistent way of implementing a service using workflows and agents.

2. Rationale, Approach and Related Work

2.1. Rationale

Overall, what we require is a set of middleware services that enable an arbitrary collection of applications to be composed (glued) and executed in a dependable manner. There are several compelling reasons why we have chosen two distinct technologies in the TCCS platform for service provisioning. We present these reasons below, after briefly summarising the main characteristics of workflow and agent systems.

Workflow is concerned with the automation of procedures where documents, information or tasks are passed between participants according to a defined set of rules to achieve, or contribute to, an overall business goal. Many organisations make use of workflow management systems for automating their business processes. A Workflow schema (workflow script) is used explicitly to represent the structure of the process in terms of tasks and temporal dependencies between tasks. A workflow schema can be regarded as a specification of a particular way of composing an application out of existing applications, with each task representing a constituent application. An application is executed by instantiating the corresponding workflow schema. Tasks (activities) are application specific units of work, and may themselves be workflows. Many of these tasks require human intervention. Workflow management systems contain specific facilities for 'work activity coordination' (mostly in the form of worklists that can be associated with workers) to enable the workers of an organisation to carry out their tasks that form the part of these workflows.

Within the context of this paper, an agent is defined as a software entity, such as a process or an active object; agents usually have some degree of autonomy and can be static or mobile. They behave according to an event/reaction model. An event is a typed data structure used for exchanging information with other agents. Whenever an agent receives an event, it executes the appropriate reaction according to the event type and its current state. A reaction is some computation acting on the agent's state and may involve generation of further events. This event/reaction model is based on a message-passing mechanism which ensures an asynchronous and anonymous communication schema. Such communication can be provided by a message oriented middleware, MOM. This kind of middleware is of particular interest when integrating applications that are not designed for simultaneous execution (loose integration). Agents act as the glue software between component applications.

We now discuss the advantages gained by integrating workflow and agent systems, bearing in mind that we want a system that will allow us to compose a given service out of existing ones as well as give us the ability to exercise dynamic control over the execution of the service.

Flexible composition: We first observe that there are two views of a service that are relevant here: *structural (static) view* that describes the configuration of a service (also referred to as the service architecture) in terms of service's constituent components and bindings between them, and the *behavioural (dynamic) view* that specifies the behaviour, indicating the coordination/interactions among the service's constituent components. Service composition therefore has two aspects, namely *structural composition* and *behaviour composition*. Modern component based programming environments (e.g. [1,2,3]) provide good facilities for structural composition, but lack facilities for behaviour composition and work activity coordination. On the other hand, workflow systems excel in providing behaviour composition and work activity coordination, but have limited facilities for structural composition. A combination of the two holds the promise of providing powerful composition capability.

Imagine that a system consists of a collection of components A, B, C,... (of types *A, B, C,…*, respectively). Each component provides services through its operations (tasks), a1, a2,...(for component type *A* and so forth). A particular composition of such operations (e.g., a serial composition, P1 = {b2; c1; a2}) represents a specific pattern of behaviour. A workflow system essentially provides facilities for behaviour composition;

P1 above represents a (workflow) process. We have chosen workflow technology for specifying and implementing service behaviour. This is primarily because we are concentrating on long running services that could be very complex in structure, containing many temporal and data-flow dependencies between their constituent services. Transactional workflows are ideally suited to dealing with such long running processes. An additional point to note is that many of the tasks of a service will require human intervention for which workflow systems provide the necessary work activity coordination support. In the case of the TCCS platform, our workflow system provides a uniform way of composing a complex task out of transactional and non-transactional tasks and supports a transactional task coordination facility to ensure that tasks get executed respecting their temporal and data-flow dependencies [4,5]; flexible worklist management facilities are also provided [6].

To illustrate structural composition, imagine composing a new type of component, *M* out of say one each of *A*, *B* and *C* and by binding them in a specific manner using connectors of the underlying middleware (e.g., RPC in case of an ORB, or asynchronous message channels in case of a MOM). We call this type of composition structural composition. The agent system of the TCCS platform uses MOM to implement reliable asynchronous message channels [7] coupled with a programming environment with facilities for composing and deploying agents (and in general, any other type of components, e.g., CORBA components) [3]. Several researchers (e.g. [8,9,10]) have argued that in the Internet/Web environment, a practical way of gluing applications is through loose coupling as provided by MOM. The main reason behind this, as stated earlier, is that asynchronous communication decouples producers of information from consumers; they do not need to be both ready for execution at the same time.

Configuration control: We concentrate on dynamic reconfiguration: ability to exercise dynamic control over the execution of a service. Like composition, we identify two aspects of dynamic reconfiguration: *behaviour reconfiguration* and *structural reconfiguration*. Behaviour reconfiguration implies that it should be possible to change the behaviour of a service by adding/deleting constituent tasks, notifications and data-flow dependencies in a consistent manner. Thus, in the example considered previously, we might wish to modify P1 from {b2; c1; a2} to {b2; a2; c3} at run time. The transactional workflow system used in the TCCS platform provides suitable ways of enforcing such changes [11]. It uses transactions ensures that changes are carried out atomically with respect to normal processing. Structural reconfiguration refers to on-line changes to service configuration that are not primarily concerned with changes to behaviour; an example being migration one or more components of a service from one node to other for the sake of load balancing. Such facilities are best provided as a part of the component based programming environment that is responsible for specifying and deploying configurations. The OLAN programming environment used by our agent system provides such facilities [12]. Because of the loose component coupling provided by asynchronous messages, structural reconfiguration is simpler to perform if components are bound together using MOM.

User support: Agents are also useful as entities that perform actions on behalf of users (user agents). In workflow management systems such user agents can aid the workers of an organisation to carry out their tasks that form the part of the workflows (e.g., assistance with email handling).

2.2. Approach

As stated earlier, by combining the two technologies, there is the real danger of confronting application designers with a bewildering choice of application development options – a situation that will almost certainly make the task of service provisioning harder rather than simpler. We have avoided this pitfall by constructing a development environment that draws heavily from the research results on *software architecture*. As we discuss below, this has enabled our development environment, called TCCS IDE (IDE: Integrated Development Environment), to provide a consistent way of implementing a service using workflows and agents.

Software architecture specification is intended to describe the structure of the components of a software system, their interrelationships, and principles and guidelines governing their design and evolution. Work in this area has produced high-level textual and graphical notations (Architecture Description Languages - ADLs) for expressing and representing architectural designs and styles [e.g., 1,2,3]. Using an ADL, the configuration of a software system (the structural, static view) is expressed in terms of components, where a component provides services to other components. A component within a system can be either a simple component, or a complex (composite) component, composed out of a group of other components. The components provide and obtain services through ports. The interaction between ports takes place via connectors; these can take one or more forms, for example, message passing, RPC, etc. depending on the facilities provided by the underlying infrastructure. ADL based programming environments provide GUI tools for developing, analysing and deploying software system configurations. In our case, OLAN [3] the system chosen by us, provides such tools for components that could be agents, CORBA objects or a combination of the two. We encapsulate workflows within components. This enables us to describe the configuration of a service in a uniform manner. At the same time, we have built tools for workflow development, analysis and deployment that closely follow the ADL approach. The overall result is that the TCCS IDE provides a consistent way of structural and behaviour composition.

Consider a simple example. Fig. 1 shows the configuration of a bank service to be made up from three components, AccountManager, Authentication and Account (we are using simplified notations of OLAN). Dark circles represent operations *provided* by a component and white circles represent the operations *required* by a component; lines connecting them represent the connectors of the underlying middleware. The AccountManager provides an operation transfer (T) that credits one account and debits another account; this component requires operations authenticate (A), debit (D) and credit (C) to implement its transfer operation. The AccountManager is a *workflow component*; the behaviour of its provided service is represented by a workflow. Fig. 1 also shows this workflow that makes use of the various operations provided by the components of the banking service (we are using a simplified version of the notations used by our workflow system). Three tasks are executed in sequence: authenticate, then debit followed by credit. For simplicity, exceptional paths dealing with situations such as failure to authenticate or insufficient funds have been omitted.

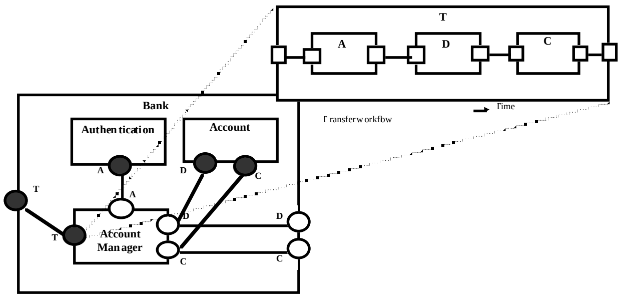

Fig.1: Software architecture and workflow

Here we see that the workflow and software configuration (software architecture) define two complementary views of a given service – the former dealing with temporal relationships between tasks to be performed, the latter providing a structural view in terms of components, bindings and services provided by the system. The workflow executes on a system by invoking the services provided by it. In essence, each task that is executed within a workflow must be supported by one or more services provided by the system. The software architecture *resources* the workflow.

Fig. 2 shows the overall architecture of our system. Interworking between the two platform enables agents to deploy and control workflows and workflow tasks to include agents. The TCCS IDE contains four main tools; underlying support for these tools is provided by the component configuration service and the workflow configuration service.

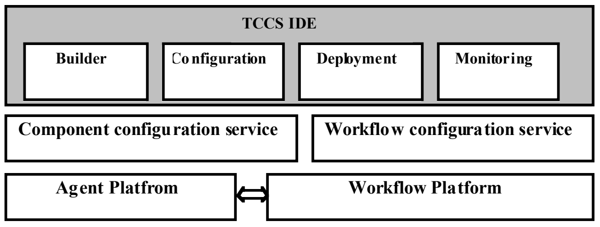

Fig.2: TCCS Platform

Each tool provides complimentary GUI based facilities for manipulating either a software configuration or a workflow. We very briefly describe these facilities of the builder, configuration and deployment tools for the case of software configurations. The builder enables creation of new component types. A number of *properties* (in the form of name, value pairs) can be associated with a component. These are used mainly for initialising the instance variables of the component. Properties are also used for specifying deployment information (such as on which node to instantiate the component). The configuration tool enables the creation of a configuration: here component interconnection is performed. The deployment tool takes a given configuration description and uses the component configuration service for creating instances of components and their inter-component bindings.

2.3. Related Work

At the start of the paper we stated several requirements a service provisioning framework should meet. Other researchers have made broadly similar observations (e.g., [13,14]). It has been pointed out that in the Internet environment, applications are frequently need to be glued 'spontaneously' for which the loose coupling as provided by MOM is ideal [9,10]. The Information Bus [8] is an early example of such a system. Our agent system provides a high level distributed programming model enabling flexible and location independent objects on top of a MOM.

Combined use of agents and workflows is being investigated by many researchers. The Agent Enhanced Workflow (AEW) system [15] uses an agent system to overcome inability of many of the commercially available workflow systems that are monolithic in structure, so unsatisfactory for use in a distributed environment. In AEW, the agent layer manages the distribution of work between disparate workflow systems. In the COSMOS system [16], mobile agents aid workflows for the purposes of contract negotiation among distributed organisations. Distinguishing aspects of our work, not emphasised in other works is the support for service composition using an ADL based environment, and, as we discuss subsequently, facilities for dynamic reconfiguration and dependability.

3. TCCS Platform Implementation

3.1. General

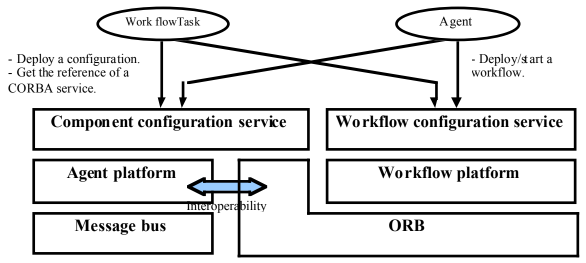
Work flowTask
Agent
- Deploy a configuration.
- Get the reference of a CORBA service.
- Deploy/start a workflow.
Component configuration service
Workflow configuration service
Agent platform
Workflow platform
Interoperability
Message bus
ORB

Fig. 3: Interoperation between agents and workflows

The TCCS platform provides interoperation between agents and workflows by means of two CORBA services: component configuration and workflow configuration. The main difficulty in interoperation is that the communication protocols used by the two platforms are different. CORBA is based on remote procedure call whereas agent communication is message oriented. This difficulty has been overcome by constructing 'proxy agents' that can straddle both platforms. Fig. 3 hints this by indicating the component configuration service extending over the ORB. The component configuration service enables a workflow task to deploy a configuration and get the reference of a deployed CORBA service. The workflow configuration service enables an agent (or any component for that matter) to deploy a workflow. This way an agent may trigger a workflow. This enables agents to act as user agents in order to manage workflows on behalf of users. We next discuss these details after summarising the main features of the two platforms, concentrating on their reliability features.

3.2. Overview of Agent and Workflow Systems

AAA Agent System: What follows is a brief overview of the AAA (Agent Anytime Anywhere) system [7,12]. One of the requirements of a glue software system is its reliability. Execution structures must be able to resist failures, in particular node failures. Agents support for this reason some transactional features : *atomicity of reactions* and *persistency*, both enabling the recovery of a consistent agent state in case of a failure. Agents are persistent. This means that the agent lifetime is not bounded to the duration of the execution. The agent state is not lost when the execution node stops or crashes. However, agent persistency is not sufficient for retrieving a consistent state after failures. Therefore, agent reactions are atomic. This property ensures that a reaction is either fully executed or not executed at all: if the reaction is fully executed, the agent new state is committed and all events triggered during the reaction are actually delivered to their destination. If a failure occurs, the system rollbacks to a consistent state by retrieving the agent's initial state and removing any events triggered during the reaction.

At the programming level, agents communicate by triggering events. At the infrastructure level, notification of events are transformed into messages. The communication mechanism is based on message queuing system, which provide asynchronous, *reliable* and *ordered communication*. The reliability property ensures that events once sent are guaranteed to be delivered in the presence of finite number of intervening machine crashes and temporary network related failures.. A recovery mechanism is provided for messages lost during network failure or node crashes. Message recovery capabilities enable lost messages to be re-sent with no involvement from the application, as soon as the communication is re-established or when a node is ready again. The infrastructure provides an additional feature, the *causal ordering of event delivery*. This property is useful for programmers because they can benefit at the application design time from an unambiguous ordering policy.

OPENflow Workflow System: The OPENflow system represents a significant departure from currently available workflow systems that are not scalable, as their structure tends to be monolithic; further, they offer little support for building fault-tolerant applications, nor can they inter-operate, as they make use of proprietary platforms and protocols. OPENflow architecture is decentralised and open: it has been designed and implemented as a set of CORBA services to run on top of a given ORB [4]. The system has been structured to provide dependability at *application level* and *system level*. Support for application level dependability has been provided through flexible task composition facility that enables an application builder to incorporate alternative tasks, compensating tasks, replacement tasks etc., within an application to deal with a variety of exceptional situations. The system provides support for system level dependability by recording inter-task dependencies in transactional shared objects (TaskControl objects) and by using transactions to implement the delivery of task outputs such that destination tasks receive their inputs despite finite number of intervening machine crashes and temporary network related failures. Such a use of transactions also provides a durable audit trail of task interactions. Thus our system naturally provides a fault-tolerant 'job scheduling' environment that maintains a durable history of application interactions.

A workflow is executed by instantiating (deploying) the corresponding workflow schema which has the effect of the creation of sets of objects that control and represent application tasks (a *TaskControl* and *Task* object per application task). At this time, the workflow execution service needs to know the location (computers) where these objects need to be created. This deployment information is specified useing the *property* mechnism mentioned in section 2.2 with respect to fig. 2, and passed on to *task and task control factories*. There can be any number of such factories in a system (see [6] which describes how workflow deployment can be performed to suit the requirements of an organisation by using such factories).

3.3. Agent - workflow interaction

Two types of agents have been provided to enable agent - workflow interaction. The first one, ProxyIn, allows an agent to provide operations as CORBA services. The second, ProxyOut, allows an agent to call CORBA services. Their implementation details have been omitted here to save space. ProxyOut agents provides a simple way of implementing workflow components (see fig. 4; in Olan ADL, a CORBA service is indicated by a square whereas an agent service is indicated by a circle and black is for provided, white for required). As the whole workflow environment is based on CORBA, a workflow is wrapped as a CORBA object (the Wfwrapper object in fig. 4). So the workflow component is actually a composite that contains a ProxyOut agent and the workflow wrapper. In this way, other agents can trigger this workflow.

The function of the workflow wrapper is to deploy the associated workflow (see fig. 5). At initialisation the workflow wrapper deploys a task factory and a task control factory. When the proxy agent receives the start notification, it calls the wrapper that loads the workflow script and invokes the workflow service. The service calls the specified task and task controller factories in order to instantiate the workflow and starts it.

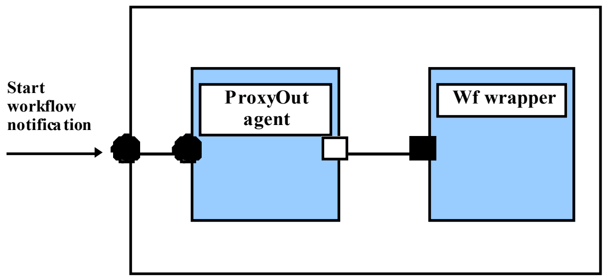

Fig. 4: Workflow composite component

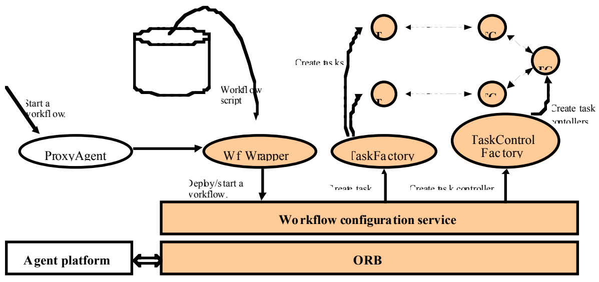

Fig. 5: An agent enacting a workflow

3.4. Support for Dynamic Reconfiguration

As stated earlier, we classify dynamic reconfiguration into structural reconfiguration and behaviour reconfiguration. The TCCS platform provides complimentary facilities for both. Olan tools enable structural reconfiguration of agent applications: adding/removing agents, changing the placement of agents and modifying the interconnection pattern [12]. Workflow tools make it possible to change the behaviour of a service by adding/deleting workflow tasks, notifications and data-flow dependencies in a consistent manner [11].

3.4.1. Agent reconfiguration

The AAA infrastructure provides a set of primitives, made available via a special agent called the *configurator*, to perform the reconfiguration process. The configurator is reliable and persistent like any other agent. The reconfiguration primitive *rebind* is used for modifying the interconnection pattern; *move* and *delete* orders allow respectively to move an agent on another node and to remove an agent from the application. 'Rebind' allows change of a reference held by an agent to another agent. In order to cope with the problem of messages in transit, the following agent rule must be satisfied:

AR1: The communication channel involved must be empty before the rebind operation can occur.

As a consequence, the communication channel related to the interconnection must be flushed. At this step, there is no message in transit anymore in the channel. The reference representing the connection can therefore be safely updated. 'Move' allows an agent to migrate to another site. In order to maintain consistency, the precondition is the following:

AR2 : All input channels of the migrating agent must be empty before the move operation can occur.

As a consequence, the basic steps to achieve the move order are the following. First, all input channels of the moving agent should be flushed. As a result, no messages are in transit anymore. Then, the target agent state is saved and recreated on the appropriate node. All references to the old agent are updated in order to point to the new agent on the new node and the old agent is destroyed.'Delete' allows an agent to be removed from an application. The precondition for this order is the following:

AR3: The agent to be removed must be isolated. This means that all channels which involve the target agent must be empty before the delete can occur.

Therefore, all input and output channels of the deleted agent should be first flushed. The target agent is then removed safely. We assume that application can run correctly without the deleted agent. The AAA infrastructure provides an operation to 'passivate' an agent. In the passive state an agent reacts to events but cannot send events.

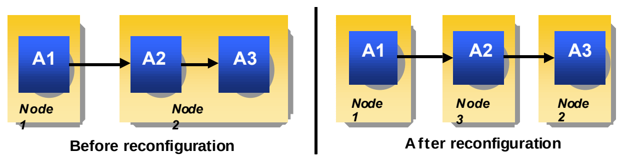

Fig. 6: Agent reconfiguration

To illustrate the reconfiguration algorithm, let us consider the following trivial example which represents the migration of agent A2 to another node (fig. 6). In order to move agent A2, AR2 must be true. This entails the 'freezing' of A2 (in the frozen state, an agent does not receive any events); so all agents which have a reference directed towards A2 must be made passive. Therefore, the configurator starts by sending a passivate order to A1 which has a reference directed towards A2; after that the move message is sent to A2. Causal ordering property ensures that this reconfiguration order will be delivered to A2 only when all messages in transit from A1 have been delivered to A2.

The reconfiguration process benefits from the agent execution model of loose component coupling. Reconfiguration algorithm does not have to take care about state preservation preserving causality and reliability. Furthermore the validation of reconfiguration actions is provided on one hand by the architecture description which specifies what kind of changes can be done, and on the other hand by the transactional event/reaction sequences which can ensure the atomicity of reconfiguration actions issued by a human administrator.

3.4.2. Workflow reconfiguration

OPENflow directly provides support for dynamic modification of workflows [11]. This is made possible because the TaskControl objects maintain the states of corresponding tasks (*waiting, active, complete*) and the inter-task dependency information of a workflow and make this information available through transactional operations for performing changes (such as addition and removal of tasks as well as addition and removal of dependencies between tasks). Workflow reconfiguration must be carried out consistently; by which we mean respecting the following two conditions:

C1: Modifications are carried out atomically (either all the changes are performed or none) with respect to the normal processing activities of the workflow.

C2: The workflow is able to execute respecting these changes.

The second condition is slightly subtle, and is discussed further. Fig. 7(a) shows a travel workflow with four constituent, *TravelPlan*, *CreditCheck*, *Flights* and Tickets (a simplified notation has been used). Tasks *CreditCheck* and *Flights* execute concurrently, but can only be started after the *TravelPlan* task has terminated and supplied the necessary data, so these two tasks have data-flow dependencies on the *TravelPlan* task. Task *Tickets* can only be started after *Flights* task has terminated and supplied the necessary data and task *CreditCheck* has notified its termination. Assume that the task Flights needs to be replaced by a task TrainJourney, with the same input-output dependencies; such a change can be performed provided the Flights task has not yet started (in state waiting). Take another example: assume that electronic payment facilities are available and, it is desirable to extend the application with a new task (Payment) with the dependencies as shown in fig. 7(b). Once these changes have been performed, the run time system should ensure that Payment task does receive its inputs. So for example, if the changes are performed after CreditCheck has terminated, its outputs should still be made available for consumption by Payment. Use of transactions to add and remove one or more tasks and to allow the addition and removal of dependencies between tasks from a running workflow ensures that changes are carried out atomically with respect to normal processing; this respects condition C1. In addition, the following restrictions need to be observed to respect condition C2 (see [11] for a additional details).

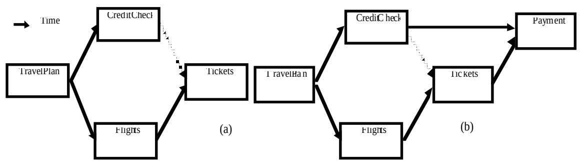

Fig. 7: Dynamic reconfiguration of a workflow

WR1: The implementation bound to a simple task can be changed, provided the task is in a wait state.

WR2: Input data sets cannot be added, removed or in anyway modified for tasks that are in state active or complete.

In summary, given that we use transactions for inter-task coordination, it is but natural to go a step beyond and use them for dynamic workflow modification.

4. An Example Application

We have implemented a demonstrator application concerned with distributed firewall management that illustrates many of the features of the service provisioning platform. Firewalls are managed by agents. Relevant data produced by these agents is used by a billing service that uses a payment workflow for charging various departments. The deployment of the entire configuration (a complicated process in reality involving deployment of hundreds of components over the network) itself is represented by a workflow.

Firewalls are security components interposed between the outside world and the internal networks of an organisation. All traffic passing through the firewall - web accesses, electronic mail, application transactions- is precisely identified, checked and allowed through or rejected depending the rules and regulations set down by the security policy. Each identification, control and security relevant action that is performed by the firewall results in the generation of an *audit record* which is collected and logged in *audit trails*. The logged information provides information about the usage of the firewall and the resources protected by it. This information is further analysed (referred to here as *traffic analysis*) in order to verify the effectiveness of the defined security policy and its implementation; and find out whether resources are misused - and if they are, who performed the misuse and how it occurred. When large number of firewalls are involved (a number between 100 to 150 within an organisation is not uncommon), it becomes necessary to perform traffic analysis in a distributed manner, as each firewall generates hundreds of megabytes of data per day.

This demonstration is distributed on 4 machines: logs are produced by the *firewalls* (represented by load generators) on the sites 1 and 2 (see fig. 8). The traffic analysis system is divided into three sets of components: two instances of the 'local analysis' on sites 1 and 2 and one instance of 'central analysis' on site 3. The local analysis filters and aggregates the records from the raw logs and the central analysis merges and further aggregates the results delivered by the local analysis and notifies the billing service when the resulting traffic analysis is available.

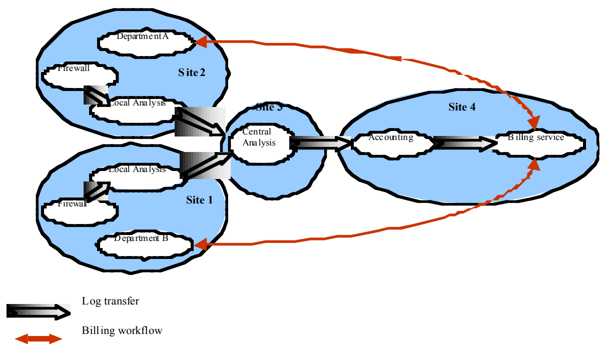

Fig. 8: Firewall Management

The configuration of local traffic analysis software system is depicted in the TCCCS IDE screen shot, fig. 9. NetWall Monitor agent is the information source which sends the records generated by the firewall to a number of agents that perform filtering. The fig. also shows the properties attached to the 'attack' filter agent.

The accounting component is in charge of creating the accounting information from the traffic analysis information produced by the central analysis component. It computes the amount of bytes sent and received by a host during a given period. When the accounting information is ready, the billing component starts the workflow in charge of coordinating the payment process for each department. The screen shot of the billing workflow component and its workflow is shown in fig. 10.

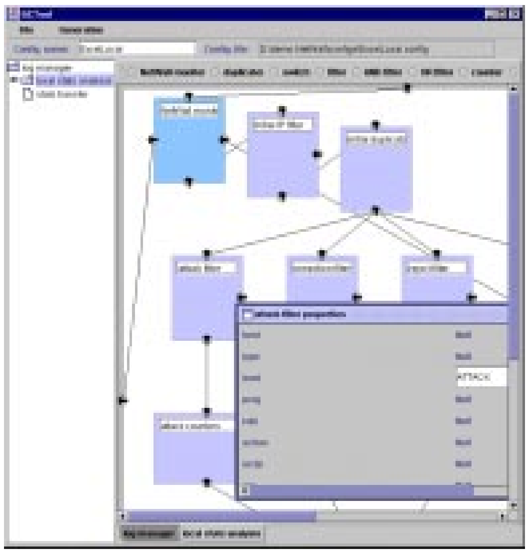

Fig. 9: Traffic analysis configuration

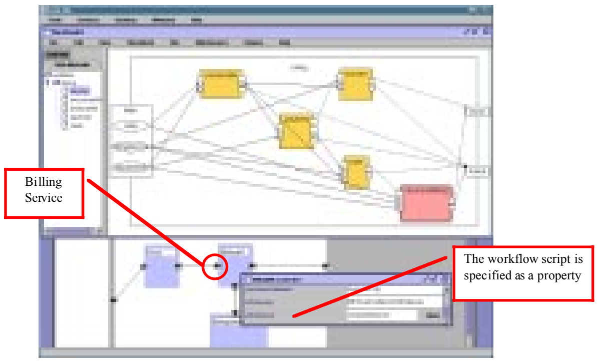

Fig. 10: Billing service workflow

5. Concluding Remarks

A service provisioning platform must provide facilities that enable an arbitrary collection of applications to be composed (glued) and executed in a dependable manner. Service composition has two aspects, namely structural composition and behaviour composition. Component based programming environments provide good facilities for structural composition, but lack facilities for behaviour composition and work activity coordination. On the other hand, workflow systems excel in providing behaviour composition and work activity coordination, but have limited facilities for structural composition. Our service provisioning platform therefore makes use of dependable workflow and agent systems. By constructing a development environment that provides a uniform way of manipulating structure and behaviour of a service, we are able to provide a consistent way of composing a service using workflows and agents. At the same time we are able to offer dynamic reconfiguration facilities for modifying the behaviour and structure of a service.

The system described here has been built. We have constructed a demonstrator application concerned with distributed firewall management that illustrates many of the features of the service provisioning platform. Firewalls are managed by agents. Relevant data produced by these agents are used by a billing service that uses a payment workflow for charging various departments. The deployment of the entire configuration (a complicated process in reality involving deployment of hundreds of components over the network) itself is represented by a workflow.

Acknowledgement

This work has been supported in part by ESPRIT LTR Project C3DS (project no. 24962). Discussions with Jeff Kramer, Jeff Magee and Valerie Issarny have been helpful. Integration of the two platforms was carried out with the help of F. Ranno.

References

[1] M. Shaw and D. Garlan, “Software architecture: perspectives on an emerging discipline”, Prentice Hall, 1996.

[2] J. Magee, J. Kramer and M.Sloman, “Regis: a constructive development environment for distributed programs”, Distributed Systems Engineering, 1(5), pp. 663-675, 1994.

[3] Balter R., Bellissard L., Boyer F., Riveill M., Vion-Dury J.-Y., “Architecturing and Configuring Distributed Applications with Olan”, Proc. Of IFIP Intl. Conference on Distributed Systems Platforms and Open Distributed Processing, Middleware 98, (N. Davies, K. Raymond, J. Seitz, eds.), Springer-Verlag, London, 1998, ISBN 1-85233-088-0.

[4] S.M. Wheater, S.K. Shrivastava and F. Ranno “A CORBA Compliant Transactional Workflow System for Internet Applications “, Proc. Of IFIP Intl. Conference on Distributed Systems Platforms and Open Distributed Processing, Middleware 98, (N. Davies, K. Raymond, J. Seitz, eds.), Springer-Verlag, London, 1998, ISBN 1-85233-088-0, pp. 3-18.

[5] F. Ranno, S.K Shrivastava and S.M. Wheater, “A Language for Specifying the Composition of Reliable Distributed Applications”, 18th IEEE Intl. Conf. on Distributed Computing Systems, ICDCS’98, Amsterdam, May 1998, pp. 534-543.

[6] J. J. Halliday, S. K. Shrivastava, S. M. Wheater, “Implementing Support for Work Activity Coordination within a Distributed Workflow System”, Proc. of 3rd IEEE/OMG International Enterprise Distributed Object Computing Conference (EDOC’99), September 27-30, 1999, University of Mannheim, Germany, pp. 116-123.

[7] L. Bellissard, N. De Palma A.Freyssinet, M. Herrmann, S. Lacourte, “An Agent Platform for Reliable Asynchronous Distributed Programming”,

[8] B. Oki, Pflueg M., Siegel A., Skeen D., “The Information Bus – An Architecture for Extensible Distributed Systems”, Operating Systems review, 27(5), PP.58-68, Dec. 1993

[9] K. Mani Chandy and Adam Rifkin Systematic Composition of Objects in Distributed Internet Applications: Processes and Sessions. Computer Journal in October 1997

[10] G. Banavar, T. Chandra, R. Strom and D. Sturman, “A case for message oriented middleware”, Proc. Of Symp. On Distributed Systems, DISC99, Bratislava, September 1999, LNCS. Vol. 1693.

[11] S K Shrivastava and S M Wheater, “Architectural Support for Dynamic Reconfiguration of distributed workflow Applications”, IEE Proceedings – Software, Vol. 145, No. 5, October 1998, pp. 155-162.

[12] N. De Palma., Bellissard L., Riveill M., “Dynamic Reconfiguration of Agent-based Applications”, *to be published in European Research Seminar on Advances in Distributed systems (ERSADS’99)*, Madeira, Portugal, April 1999.

[13] Y. Hoffner, C. Facciorusso, S. Field and A. Schade, “Distribution issues in the design of and implementation of a virtual market place”, Distributed Applications and Interoperable Systems, Ed. Lea Kutvonen, Hartmut Konig, Martti Tienari, Kluwer Academic Publishers, 1999, ISBN 0-7923-8527-6, pp. 405-422.

[14] R. Bartell et al, “The MediaXact system - a framework for personalised electronic commerce services”, Bell Labs Technical Journal, April-June 1999, pp. 153-173.

[15] D. F. Judge, B. Odgers, J. Shepherdson and Z. Li, “Agent enhnced workflow”, BT Technical Journal, 16:3, 1998, pp. 79-85.

[16] F. Griffel, M. Boger, H. Weinreich, W. Lamersdorf and M. Merz, “Electronic contracting with COSMOS - how to establish and execute electronic contracts on the Internet”, Proc. Of IEEE/OMG Second Enterprise Distributed Object Computing Workshop (EDOC’98), La Jolla, CA, November 1998.