# Workflow: A Language for Composing Web Services

Giacomo Piccinelli[1], Scott Lane Williams[2]

[1] Hewlett-Packard Laboratories, Stoke Gifford Park
BS34 8QZ Bristol, United Kingdom
Giacomo_Piccinelli@HP.com

[2] Hewlett-Packard Software & Solutions, 11000 Wolf Road
95014-0691 Cupertino (CA), USA
Scott_L_Williams@HP.com

**Abstract.** The introduction of Web Services has a profound impact on component models. The interaction processes behind a service become integral part of the component type, and as such formally described and automatically manageable. Workflow emerges as the reference model for the description of interaction processes associated to individual web services. In the DySCo (Dynamic Service Composition) project, we investigate the use of workflow for both the modelling and implementation of composite solutions based on web services. Key aspect of DySCo is the separation between composition and coordination logic. In this paper, we discuss the composition model defined in DySCo, and a technology framework to enforce it.

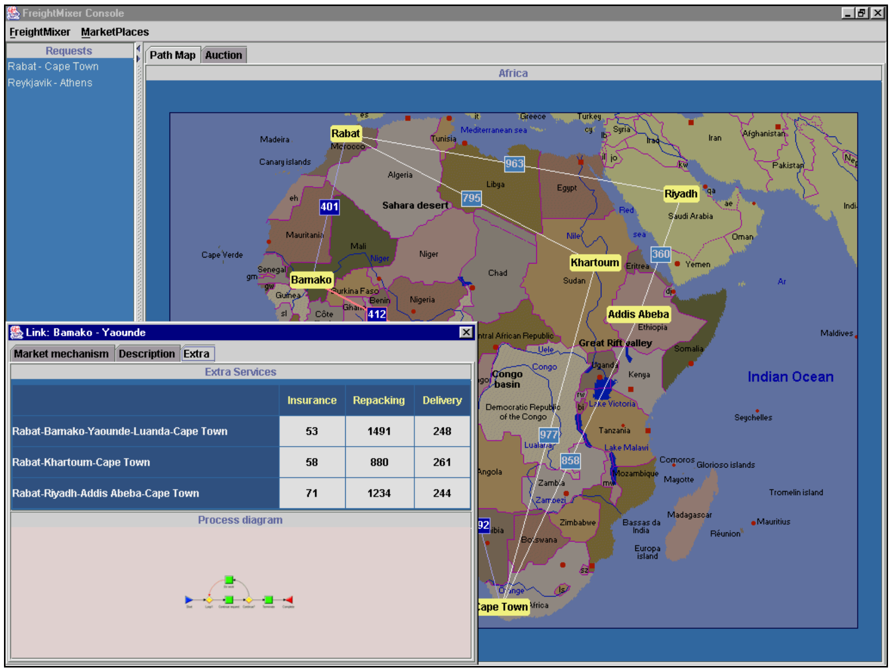
## 1 Introduction

Since their early definition, composition has been a central concept for web services. From a business perspective, web services represent a new channel for the offer as well as the acquisition of business services. Peculiarity of the web service channel is that the interaction process between service provider and service consumer is completely automated. Beyond electronic data transfer, automation extends to all aspects of business interaction. Negotiation of service delivery terms and management of service-level agreements are just some examples. A comprehensive description of the foundational and operational aspects of the web service model can be found in [5,14].

In the web service model, the provider of a new web service WS drives the composition of internal and external capabilities in order to produce a new capability. Both internal and external capabilities are modelled as web services that act as service components for WS. Similarly, WS can be used as service component for other web services. The fact that a capability is available internally or needs to be acquired externally reflects on the design of a service. Still, the web service model enforces a clear separation between a service component and the related service provider. Different providers can be used for the same capability under different circumstances, and the selection logic is integral part of the overall design of a web service.

The decupling of web service and web service provider is closely related to the most noticeable feature of the web service model: process-oriented interfaces. In traditional component models, method signatures represent the only information on the interaction requirements of a component. While the invocation of a method can trigger complex interaction processes, interaction logic is not formally exposed. In the web service model, the interaction processes associated to a web service are explicitly formalized and exposed. WSCL (Web Service Conversation Language) [1] and WSFL (Web Service Flow Language) [8] are examples of languages for the formalization of interaction processes associated to web services. The rationale for process-oriented interfaces is composition in general, and dynamic composition in particular. The composition logic for a web service depends on the interaction processes of the service components for their orchestration. Interaction processes are central to the selection logic for service providers.

The objective of the DySCo (Dynamic Service Composition) [11] project is the development of a modelling and technology framework for the dynamic composition of web services. The distinctive element for DySCo with respect to other approaches [2,5,8,17] is the separation between composition logic and coordination logic. The designer can concentrate on the coordination of business capabilities, which are modelled as business roles. The coordination logic for the service providers is derived automatically based on the roles that each provider commits to play. In DySCo as well as in the wider web service community, workflow [4,6] is the reference model for the formalization of both interaction logic and composition logic of web services.



| | Insurance | Repacking | Delivery |
|---|---|---|---|
| Rabat-Bamako-Yaounde-Luanda-Cape Town | 53 | 1491 | 248 |
| Rabat-Khartoum-Cape Town | 58 | 880 | 261 |
| Rabat-Riyadh-Addis Abeba-Cape Town | 71 | 1234 | 244 |

**Fig. 1.** Monitoring console in the prototype for the service composition system of FreightMixer

In this paper, we first introduce (Section 2) a reference scenario for service composition based on the FreightMixer prototype. In (Section 3), we discuss some of the requirements for web service composition, with special focus on the dynamic aspects of composition. In (Section 4), we describe the composition model adopted in DySCo. In (Section 5), we present an overview of technology in the DySCo framework. In (Section 6), we report on related works. In (Section 7), we present some conclusions and remarks drawn from our involvement in DySCo.

## 2 Service Composition in FreightMixer

FreightMixer is a provider of freight services. Peculiarity of FreightMixer is a business model entirely based on web services. The company is not a real one, but it represents a realistic scenario for the use of web services. The scenario provides a conceptual framework that can be applied to a wide range of business contexts. More details can be found in [11]. In this section we summarize the service composition model used by FreightMixer.

The freight market is highly competitive, and customers explore multiple options before every shipment. Customers send RFQs (request for quote) to specific providers. Alternatively, customers can use auction-based instruments made available by electronic marketplaces. Switching costs are negligible, and the best offer wins the deal. Service providers are under pressure for competitive pricing of comparable levels of service. In terms of service execution, customer’s main concern is that service-level agreements are respected (e.g. level of notification). The main concern for service providers is the cost of the execution infrastructure. Fixed costs are quite significant, and full utilization of resources is crucial. For example, the cost of shipping a container is almost independent from the content. Maximum usage of capacity translates into lower costs per unit.

FreightMixer offers an end-to-end freight service without owning fixed transport or transport-related infrastructure. When a new service request arrives, FreightMixer acquires services from other providers and combines them into a complete package. The result of the combined services constitutes the infrastructure upon which FreightMixer bases its offer. The composite nature of FreightMixer’s offer is completely transparent to the customer. In terms of service execution, the providers for some of the service components may interact directly with the customer. Still, the interaction is on behalf and under the responsibility of FreightMixer. Providers of service components may also interact with each other. The interaction between service providers as well as with the customer depends on the overall composition logic decided by FreightMixer, and is regulated by the agreements between FreightMixer and each individual service provider.

In (Fig. 1) is represented the monitoring console for a prototype system that implements FreightMixer's operational model. The panel on the left contains the list of the service requests for which a solution is currently under development. Focusing on a specific service request (e.g. sending goods from Rabat to Cape Town), the main panel presents the various options under consideration. For each leg of each route, the system presents the results of the ongoing negotiations. Expanding one of the legs, the system shows the ongoing negotiations for the specific leg. The system shows the various negotiation parameters, and in particular the service deliver process required to any service provider that wants to cover that leg. Service delivery processes are fundamental for service composition. Individual processes reflect the requirements for the provider of a leg of transport, the provider of the previous leg, the provider of the next leg, and the providers of corollary services such as insurance. Processes also reflect monitoring requirements from FreightMixer, and special requirements from the customer (e.g. notification). While not represented in the picture, the system simultaneously manages the negotiation with the customer. The system also monitors the execution of the complete solution if the deal is won. More details can be found in [11,13].

The core value of FreightMixer lays in the capability to aggregate the appropriate mix of service providers around a single objective. FreightMixer exploits competitive offer from other providers for the implementation of a solution matching the needs of its customers. The success of FreightMixer depends on the ability to acquire spare capacity at low cost, and to efficiently combine different resources. Service composition and negotiation are the pillars of FreightMixer's business model, and web services play a key role in their automation.

## 3 Requirements for Dynamic Composition

The concept of dynamic composition [10,16] is subject to different characterizations within the web service community. One aspect of web service composition is the binding between formal and actual provider for a service [14]. The set of service providers involved in an instance of a composite web service can be dynamically populated, hence the reference to dynamic composition. To avoid ambiguity, we refer to such aspect of composition as aggregation. Explicit management of aggregation logic is an important requirement for dynamic composition.

A second aspect of dynamic composition is the coordination of different web services [18]. The mainstream approach is to have a single entity responsible for coordination, hence the reference to coordination as orchestration. While agreeing on the need for a centre of responsibility in a composite service, our view on the orchestration model is that more can be added in order to capture the full potential for web service composition. In particular, we refer to peer-to-peer interaction and delegation. Using the FreightMixer example, the providers of two legs of transport route may need to interact with each other directly in the context of the overall service created by FreightMixer. We propose both delegation and peer-to-peer interaction as key requirements for dynamic composition.

Considering the application domain for web services, business applications represent the most important source of requirements for web services in general and composition in particular. The close relation between business services and web services has strong implications on web service solutions. Business processes are the driving force behind business services, and the association of business processes with workflow models and technology [4] has deep roots in the business world. Leveraging the conceptual framework and technology associated with workflow is a general requirement for web services. In the specific case of composition, workflow represents an important source of opportunities as well as requirements. Coordination of different capabilities towards the achievement of a specific objective is a fundamental concept in the workflow model. A mapping between workflow and web services can immediately leverage existing expertise on the business side as well as on the technical side. On the technical side, integration between web service and workflow technology can provide an important integration route towards existing business applications.

## 4 Web Service Composition in DySCo

The objective for DySCo can be summarized as a composition framework for web services that match the requirements of a company such as FreightMixer. Initial results on the composition methodology developed in DySCo can be found in [11]. The aggregation aspects of the methodology are described in [13]. In this paper, we discuss the composition model defined in DySCo. The focus is on the separation enforced by the model between composition and coordination logic.

The composition model defined in DySCo is based on the workflow model. Both composition and coordination logics are specified as workflows. The designer of the web service defines the composition logic once. Specific tools in the DySCo framework automatically derive the coordination logic required for different aggregation patterns. Delegation and peer-to-peer interaction are integral part of both the composition and the coordination models.

In the remaining part of this section, we first give an overview of the workflow model (Section 4.1). We then describe in more detail the composition (Section 4.2) and coordination (Section 4.3) models defined in the DySCo framework.

**Table 1.** Basic grammar for workflow processes. N is a set of variable names. V is a set of constant names (values). R is a set of resource names. T is a set of task names. The condition *c* in the *choice* is a binary function with domain N.

$$
\begin{array}{lll}
\mathbf{W ::= \varepsilon} & \textit{empty process} & \\
\quad \mid \mathbf{t_r(\sigma)} & \textit{task} & (\sigma : \mathcal{N} \to \mathcal{V} \quad r \in \mathcal{R} \quad t \in \mathcal{T}) \\
\quad \mid \mathbf{W.W} & \textit{sequence} & \\
\quad \mid \mathbf{W +_c W} & \textit{choice} & \\
\quad \mid \mathbf{W \parallel W} & \textit{concurrency} & \\
\quad \mid \mathbf{!\,W} & \textit{loop} &
\end{array}
$$

**Table 2.** Labelled Transition System (LTS) defining the process evolution function $\rightarrow$ in the algebra (W, $\rightarrow$)

$$\textbf{(step)}\quad \Omega::t_r(\sigma) \xrightarrow{\varphi(t,r,\sigma')} \Omega \lhd \sigma'::\varepsilon \qquad \text{where } \sigma' = \rho\,(t, r, \Omega \rhd \sigma)$$

$$\textbf{(loop)}\quad \Omega::!W \xrightarrow{\tau} \Omega::W.(!W)$$

$$\textbf{(seq1)}\quad \frac{\Omega::W1 \xrightarrow{\alpha} \Omega'::W1'}{\Omega::W1.\,W2 \xrightarrow{\alpha} \Omega'::W1'\,.\,W2}$$

$$\textbf{(seq2)}\quad \Omega::\varepsilon\,.\,W2 \xrightarrow{\tau} \Omega::W2$$

$$\textbf{(choice)}\quad \Omega::W1 +_c W2 \xrightarrow{\tau} \Omega::WX \quad \textit{where } X = \mathrm{eval}(c) \in \{1,2\}$$

$$\textbf{(comp1)}\quad \frac{\Omega::W1 \xrightarrow{\alpha} \Omega'::W1'}{\Omega::W1 \parallel W2 \xrightarrow{\alpha} \Omega'::W1' \parallel W2}$$

$$\textbf{(comp2)}\quad \frac{\Omega::W2 \xrightarrow{\alpha} \Omega'::W2'}{\Omega::W1 \parallel W2 \xrightarrow{\alpha} \Omega'::W1 \parallel W2'}$$

$$\textbf{(comp3)}\quad \frac{\Omega::W1 \xrightarrow{\alpha} \Omega'::W1' \qquad \Omega::W2 \xrightarrow{\beta} \Omega''::W2'}{\Omega::W1 \parallel W2 \xrightarrow{\mu(\alpha,\beta)} \Omega' \bigtriangledown \Omega''::W1' \parallel W2'}$$

## 4.1 The Workflow Model

The objective of workflow [6] is to model and manage the coordination of a set of resources towards the achievement of a given result. Resources are modelled as independent and self-contained units that have the capability to perform specific tasks. While resources may be capable of autonomous behaviour, the tasks specified in a workflow are performed only upon request. The coordination logic in the workflow model is based on an orchestration approach. A single logical entity is in charge of maintaining the state of the process, and to request the resources to perform tasks. An algebraic characterisation (W, $\rightarrow$) for workflow processes is given in (Tab. 1 and Tab. 2).

The atomic element in the specification of a workflow process (Tab. 1) is the task. A task is defined by a name that indicates the activity required to a resource, as well as a name that indicates the resource that has to perform the activity. The specification for a task includes indications on the information that is supplied to a resource, and on the information that the resource will produce as a result of the activity. The flow logic for the process is expressed (Tab. 1) by the sequence, choice, concurrent, and loop operators. An operational definition for the execution semantics of a workflow process is described by the LTS in (Tab. 2).

The execution semantics for a workflow process is based on the concept of a global state $\Omega$ that contains a selection of the information generated during the execution of the process. The execution of a task (Tab. 2 - step) relies on a function ($w$) to extract form $\Omega$ the information required by the resource. In the same way, a function ($v$) feeds into $\Omega$ new information generated by the resource. Only one resource is involved in each task ($r \in R$). Transactional access to $\Omega$ is not explicit in the basic workflow model. The execution semantics also defines a concept of visibility over the evolution of the process. In the case of a task, a function $\varphi$ defines the information to make visible based on the task name, the resource name, and the information involved in the execution of the task itself.

A detailed discussion of the theoretical framework for workflow is outside the scope of this paper, and more information can be found in [7,9]. The reason why we focus our description on tasks more than on flow control will become apparent in the next section.

## 4.2 The DySCo Composition Model

One of the main limitations that prevent the traditional workflow model from fully exploiting the compositional potential of web service is the focus on activities. The unit of process in a traditional workflow revolves around asking a resource to perform an activity (task). The result from one task then constitutes the input for other tasks. The resource involved in a task is not aware of the resources involved in other tasks. The interaction between different resources is mediated by the process manager, which acts as logical point of concentration for the data flow associated to the process.

The composition model we propose for web services retains the overall structure of traditional workflow (Tab. 1 and Tab. 2), with exception of the task. In DySCo the focus shifts from activity to interaction. Resources become roles, and the task becomes an interaction step. In an interaction step, sets of roles interact towards the achievement of a given objective. In traditional workflow, the task name contains sufficient information for the resource to understand precisely the activity to perform. In an interactive step, the step name contains sufficient information for the roles involved to understand precisely the way in which they have to interact with each other. As in traditional workflow, information from the overall state of the process may be provided as input for the execution of an interactive step. An interactive step may produce output information that contributes to the state of the overall process.

Given the set $R$ of role names and the set $S$ of names of interactive steps, the entry for the task in the process grammar of (Tab.1) becomes:

$$s_r(\sigma) \quad \textit{interactive step} \quad (\, \sigma : N \rightarrow V \quad r \subseteq R \quad s \in S \,)$$

The entry (step) for the process evolution function in the LTS of (Tab. 2) is replaced by:

$$\textbf{(int-step)} \;\; \Omega{::}s_r(\sigma) \xrightarrow{\lambda\,(t,\, v,\, \sigma')} \Omega \, v \, \sigma'{::}\varepsilon \qquad \text{where } \; \sigma' = \rho\,(s, r, \Omega w \sigma) \quad r \subseteq R \quad v \subseteq R$$

The involvement of multiple roles in an interactive step affects both the output function $\rho$ and the visibility function $\varphi$ for the step. The output function takes into consideration the contributions from all the roles involved in the step. The visibility function can give different perspectives on the execution of the interactive step depending on the set of roles defined by the observer. The granularity as well as the complexity of the interaction can vary between steps, but interactive steps maintain the atomicity property typical of tasks in traditional workflow.

The use of roles enforces a level of indirection between capabilities and capability providers. In particular, the interaction between the multiple roles in an interaction step is independent from the providers that will cover each role. Similarly to resource names in traditional workflow, role names are used consistently across different steps in one process. The equivalent of a task in traditional workflow can be obtained with an interactive step that specifies only one role. Similar conditions can also be created in the aggregation phase if one provider covers all the roles in an interactive step.

Interactive steps and direct interaction between roles enable explicit modelling of peer-to-peer interaction, as well as delegation. Peer-to-peer interaction and delegation constitute the basis for both the coordination and the aggregation models adopted in DySCo.

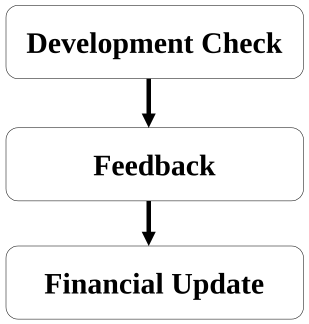


**Fig. 2.** Section of process modelled by FreightMixer. The customer first verifies the stage of delivery. There is then an exchange of information between customer and transport provider. The final step involves the interaction between customer and a financial institution.

## 4.3 The DySCo Coordination Model

The coordination model adopted in DySCo operates at role level, and is entirely based on peer-to-peer interaction. The approach is to derive from a DySCo process D a set of traditional workflow pr ocesses $\{P_j\}$, such that the result of the combination of all the $P_j$ is equivalent to D. A detailed description of the equivalence relation is outside the scope of the paper, but the general idea is that it is based on the classic notion of bisimulation [9]. The visible aspects of process execution are modelled by functions such as $\varphi$ (Tab.1) and $\lambda$ (int-step).

Given a DySCo process D and the set R of all the roles involved in D, let us consider P(R) the set of all the possible subsets of R. Let us also consider C(P(R)) the set of all the subsets of P(R), such that $c \in$ C(P(R)) implies that the union of all the sets in $c$ is equal to R. Given a set $c = \{c_j\} \in$ C(P(R)) the coordination model defines a set $W = \{W_j\}$ of traditional workflow processes such that there is a one-to-one relation between the elements of $c$ and $W$. Each process $W_j$ contains the orchestration logic for the web services of the provider covering the roles in the corresponding $c_j$, as well as the interaction and synchronisation logic with other providers. The interaction logic in $W_j$ refers to the exchange of service-level information between providers. Synchronisation logic refers to the signalling between providers required in order to align the global flow of execution.

For example, FreightMixer may have a DySCo process F (Fig. 2) related to a service offer based on frequent customer updates. One of the steps in the process can be *Development Check*, which specifies the gathering of information by some roles (e.g. *customer*) from other roles (e.g. *transport provider* and *financial institution*). For the entity covering the role *transport provider*, the $W_t$ related to F describes the usage logic for internal service capabilities (e.g. to generate a report). $W_t$ also indicates to send the report to *customer*, or to expect *customer* to potentially ask for it. Copy of the report is sent also to the *financial institution*. The $W_c$ for the *customer* specifies to wait for a communication from the *transport provider*, or the option to ask for the report. The next step in F is *Feedback*, which specifies an exchange of validation messages between some roles (e.g. *customer*) and other roles (e.g. only to the *transport provider*). $W_c$ and $W_t$ capture the activity required to the related entities. $W_c$, $W_t$, and $W_f$ (financial institution) also include synchronisation mechanisms such that the financial institution does not start the activities involved in *Financial Update* until *Feedback* is completed.

The coordination model preserves the level of indirection introduced by the composition model between a capability (captured by a role) and the web services that implement the capability. Such indirection has direct consequences on delegation as well as aggregation. In terms of delegation, the main implication is that trust models for service providers can be based on the roles they play in the composite solution, in addition to the web services they offer. In terms of aggregation, the shift is from the provision of sets of web services to the fulfilment of a specific role. The impact on models and techniques for negotiation are described in [13].

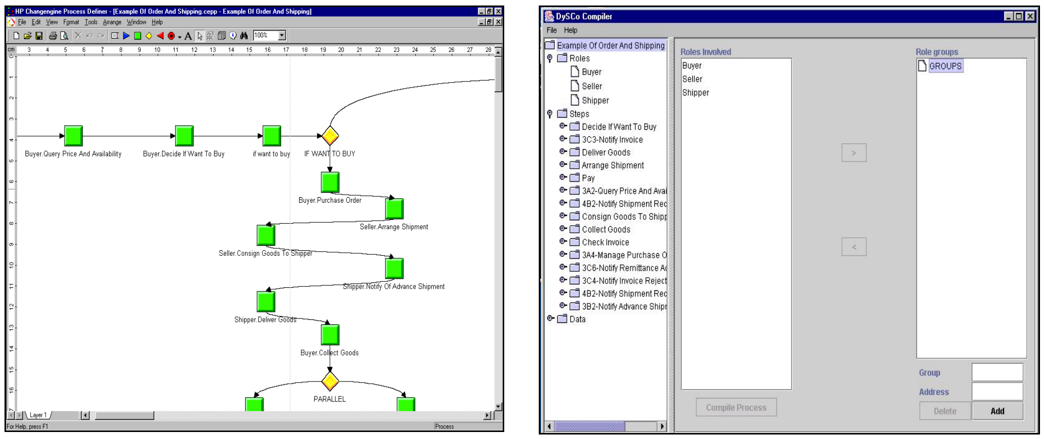


**Fig. 3.** Two components of the DySCo implementation framework. On the left, the design environment for DySCo processes. On the right, the projection generator (view on the creation of role groups).

## 5 The DySCo Implementation Framework

The DySCo implementation framework includes development tools (Fig. 3) as well as execution infrastructure to support the complete lifecycle of a composite web service. In this section, we describe the part of the framework related to the definition of the composition logic for the web service. We also describe the components involved in the automatic generation of the coordination logic for specific configurations of role distributions.

The composition logic for a web service is described using a graphical environment (Fig. 3, left) based on an extension of the commercial tool iGrafx™. The process model derives from the reference standard for workflow processes specified by the WfMC (Workflow Management Coalition) [6]. The design environment for DySCo covers processes as well as steps. In addition to the flow operators in the basic workflow model (Tab. 1), specific design patterns allow the use of higher-level constructs. Multiple branching and conditional loops are examples of such constructs. The use of sub processes is also possible. Concerning interactive steps, the interaction logic for the roles is based on an abstract execution environment also defined in DySCo. The abstract execution environment includes concepts such as a virtual repository shared by the roles, as well as direct role-to-role interaction (see [11] for more detail). The specification of the steps is based on workflow. The design environment is the same for both processes and steps. Libraries can be created for processes as well as steps. As an example of library for interactive steps, we captured and collected in one library all the currently specified PIPs (Partner Interaction Processes) specified in the RosettaNet standard [12,15].

The composition logic for the web service involves multiple roles. The coordination logic for the actual providers (and consumers) involved in the different instances of the web service depends on the roles that each party covers. The projection generation environment (Fig. 3, right) allows the automatic generation of the coordination logic for different configurations of role groups. When the files containing the composition logic are loaded into the projection generator, the tool automatically derives all the roles involved. It is then possible to specify a number of group names, and the association between roles and groups. Once all the roles have been assigned to at least one group, the tool generates for each group the workflow processes describing the related coordination logic. The activity can be repeated for different role groups. As specified in the DySCo coordination model, the projection generator adds synchronisation operations to the service logic required to the roles in a group. The control over the algorithms used for the creation of the projections makes possible formal proofs of equivalence between projections and the process from which they derive.

## 6 Related Works

XLANG [17] (led by Microsoft) and WSFL [8] (led by IBM) are the main representatives of a recent stream of activities promoting workflow as the reference model for the composition of web services.

Both initiatives focus on the internal aspects of a composite solution. WSCL [1] (led by HP) is the main representative of the complementary stream of activities promoting workflow as a model for the external aspects of web services. In terms of technology platforms, almost every commercial product for workflow management currently supports or has plans to support web services. E-Flow [2] represents a concrete example of composition platform for web services.

Currently focused on the more general problem of business-to-business (B2B) integration, the work of organisations such as ebXML [3] and RosettaNet [15] plays a fundamental role for web services. The creation of a common ontology including dictionaries as well as processes is a prerequisite for an effective automation of composition. The W3C organisation [5] acts as aggregation point for the work on architectures, protocols, and description models for web services.

## 7 Conclusions

While composition in general is at the centre of the web service model, the application context for web services sets specific requirements on composition models. Business applications and business services represent the main application domain for web services, and workflow represents the most established conceptual framework for the composition of business capabilities. We propose that workflow can play a central role for web services, both in terms of model and technology.

In the DySCo (Dynamic Service Composition) project, the workflow model becomes the basis for a multi-layered composition framework for web services. The distinctive aspect of DySCo is the separation of composition logic from coordination and aggregation logic. The model enforces a level of indirection between the capabilities involved in a composite solution and the configuration of web services that deliver such capabilities. The implementation framework for DySCo makes explicit reuse of workflow technology for the modelling and enactment of composite solutions.

## References

1. Banerji A., and others: Web Services Conversation Language (WSCL) 1.0. W3C Note (2002)
2. Casati F., and Shan M.C.: Dynamic and Adaptive Composition of E-Services. In: Journal of Information Systems, Vol. 6 No. 3 (2001)
3. ebXML: Reference Web Site for the Organization. http://www.ebXML.org (2002)
4. Georgakopoulos D., Hornick M.F., Sheth A.P.: An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure. In: Distributed and Parallel Databases Vol. 3 No. 2 (1995)
5. Haas H., and others: Web Services Activity. W3C Working Groups, http://www.w3.org/2002/ws (2002)
6. Holligsworth, D.: The Workflow Reference Model. Workflow Management Coalition (WfMC) (1994)
7. Hoare C.A.R.: Communicating Sequential Processes. In: Communication of the ACM, Vol. 21 No. 8 (1978)
8. Leymann, F.: Web Services Flow Language (WSFL 1.0). IBM (2002)
9. Milner R.: Communication and Concurrency. Prentice-Hall (1989)
10. Nierstrasz, O., and Meijler, T. D.: Requirements for a Composition Language. In: Ciancarini, P., Nierstrasz, O., Yonezawa, A. (Eds.): Object-Based Models and Languages for Concurrent Systems, LNCS 924 (1995)
11. Piccinelli G., Di Vitantonio G., and Mokrushin L.: Dynamic Service Aggregation in Electronic Marketplaces" In: Computer Networks Journal, Vol. 37 No. 2, Elsevier Science (2001)
12. Piccinelli G., Finkelstein A., Stammers E.: Automated Engineering of e-Business Processes: the RosettaNet Case Study. In: Proc. 6th Int. Conf. on Systemic, Cybernetics, and Informatics, Orlando, Florida, USA (2002)
13. Piccinelli G., Preist C., and Bartolini C.: E-Service Composition: Supporting Dynamic Definition of Process-oriented Negotiation Parameters. In: Proc. IEEE 2nd e-Negotiations Workshop, Munich, Germany (2001)
14. Kuno H.: Surveying the E-Services Technical Landscape. In: Proc. 2nd Int. Workshop on Advance Issues of E-Commerce and Web-Based Information Systems (WECWIS). Milpitas, California, USA (2000)
15. RosettaNet: Reference Web Site for the Organization. http://www.RosettaNet.org (2002)
16. Seaborne A., Stammers E., Casati F., Piccinelli G., and Shan M.: A framework for business composition. In: Proc. W3C Workshop on Web Services, San Jose, CA, USA (2001)
17. Thatte S.: XLANG – Web Services for Business Process Design. Microsoft (2001)
18. Stearns M. and Piccinelli G.: Managing Interaction Concerns in Web-Service Systems. In: Proc. 2nd IEEE Int. Workshop on Aspect Oriented Programming for Distributed Computing Systems (AOPDCS), Vienna, Austria (2002)