AN ODP APPROACH TO THE DEVELOPMENT OF LARGE MIDDLEWARE SYSTEMS

Peter F. Linington

Computing Laboratory,
University of Kent at Canterbury
Kent CT2 7NF, UK

pfl@ukc.ac.uk

Abstract: Since the Reference Model for Open Distributed Processing was completed, work in ISO in this area has concentrated on the definition of a number of supporting standards to add detail to the basic framework. Taken together, these provide a powerful structure for the support of large federated systems and provide a basis for the enhancement of tools for the development and maintenance of large middleware systems.

This paper describes the main features of the new work and speculates on how it can be applied to augment the tools used to design and manage such systems, and, by so doing, can increase their flexibility.

Keywords: Distributed processing, ODP

1 INTRODUCTION

This paper reviews the problems of designing, implementing and managing large distributed systems, reports on the recent ODP work that are directed at different parts of the problem and suggests a direction which middleware developments may take to reduce the costs of providing large federated systems.

Large distributed systems are often long-lived and span organizational boundaries. They need to evolve to meet changing requirements over a long period of time, and are typically supported by loosely coordinated management and maintenance teams with divided responsibilities and objectives. Thus the system is likely to fall into a number of distinct management domains and any coordinating authority will operate

L. Kutvonen et al. (eds.), *Distributed Applications and Interoperable Systems II*

on the basis of a fairly high level view which is imperfectly communicated to the individual domains. In other words, large distributed systems generally have a federated structure and so their development must pay particular attention to the problems of federation.

The Reference Model for Open Distributed Processing (RM-ODP) was created within the International Standards Organization during the early 1990's in order provide a stable framework for a broad family of standards for middleware and other related technologies [1][2]. Recent work within ISO has concentrated on more detailed elaboration of key parts of the framework and standardization of critical components needed to ease the federation of independently developed systems. The focus for specification of the basic middleware technologies themselves has shifted into shorter term, industry-based consortia such as the Object Management Group, although there is continuing active liaison to ensure that a coherent technical solution results. A selection of the most important standards is published by both organizations, and others are related by cross-reference.

The RM-ODP recognizes that the design and specification of any significant distributed system is a complex activity, bringing together the work of a wide range of experts. The activity can be divided into a number of areas of concern, and a designer considering one part of the problem does not necessarily have to be aware of the full detail being worked out in other areas. The different parts of the design do, however, need to be consistent, and detailed design at the component level must not defeat policies and objectives set out for the system as a whole. The RM-ODP supports the separation of concerns by introducing the idea of there being a number of viewpoints, each of which is specialized to support one aspect of the design process. The different viewpoints are inter-related by stating a series of correspondences between terms in each pair, and by establishing correspondences between the interpretations of the different languages in which the viewpoints are expressed.

The underlying vision behind this model is one of an increasingly mechanized and integrated development environment. System building and system management tools will draw upon different parts of the various viewpoints, guaranteeing, for example, that modification of the system's configuration to extend its function does not conflict with an established system-wide security or accounting policy. A simple precursor of this kind of division of the system into related parts with differing scopes and lifetimes is already familiar in the division between interface definition languages and object implementation languages. Others current examples can be found in the separation of application design from middleware configuration and feature selection.

Realizing this vision requires the components of the development environment to be designed so as to exploit information from a variety of sources, even if their use falls primarily into a particular viewpoint. Tools need to be able to play their part in validating the complete set of specifications, and in giving users meaningful feedback on errors and inconsistencies that may arise from interaction of the various specification elements.

The RM-ODP defines five viewpoints. They are the enterprise, information, computational, engineering and technology viewpoints. The different areas of concern they address can be summarized as follows:

1. The enterprise viewpoint is concerned with establishing the environment in which the system is to operate. It includes both those aspects of the organizational structure and objectives that need to be interpreted or referenced within the remainder of the specification and the system-wide policies which are intended to control the system's design and operation;
2. The information viewpoint brings together the shared view of the meaning of and constraints on key information elements, and, in so doing, gives the basis for system-wide consistency. It is the information viewpoint that provides the common interpretation guaranteeing that a concept identified in a user interface and a concept referenced in a remote invocation of an interface operation are the same concept;
3. The computational viewpoint is the main focus for functional design. The computational language defines an abstract object model – the virtual machine that interprets the computational design and thus has to be realized by any supporting middleware. The computational design identified interactions between objects at interfaces, and these interactions will need to be supported either by communication between systems or by locally optimized interactions, depending on the way the system's functions are to be distributed at any particular moment;
4. The engineering viewpoint must define the interpreter for the computational model; it consists of a series of templates for the computational interactions, parameterized so as to support a range of different policies selected either for the enterprise, or on a finer scale. The RM-ODP supports this parameterization by defining a series of transparencies, representing requirements that particular common problems (such as a lack of migration transparency) need to be solved;
5. The technology viewpoint returns to the specification of boundary conditions on the design, this time concentrating on the enumeration of standard technologies on which the design is to be based; it is primarily a catalogue of references to existing standards used by the system's designers.

2 ENTERPRISE MODELLING

The RM-ODP itself gives only rather general information about the form of the enterprise language. This is because it is intended to provide a general-purpose framework, applicable to very many different situations and organizational structures, and specific fixed constraints would, therefore, be likely to restrict its scope. Work is now in progress within ISO to create an additional, more detailed enterprise framework able to express the constraints applicable to systems used in a wide range of enterprise structures, and this work is making the enterprise language much more specific [3].

An enterprise specification is object based. It is structured by defining a number of communities, each of which is formed from enterprise objects to meet a stated objective, and which, taken together, capture the aims and purpose of the organizations to be supported by the system of interest. Communities have types, and the type is expressed in terms of community behaviour, giving the interactions of a number of roles. These roles are essentially the parameters of the community type

and are filled by suitable objects, each capable of satisfying the corresponding role requirements when a community is formed. Each object can fill a number of roles in one or more communities, and a number of roles can be filled by a single object. However, there may be constraints requiring that related roles in coupled communities be filled by the same object, or that particular roles in a single community be filled by different objects (for example, it may be stated that an auditor cannot be a financial decision maker within the same organization).

Communities are configurations of objects, and, as such, they are themselves objects. It is thus possible to nest communities, with one community filling a role in some larger, enveloping community. Combination of communities is not restricted to this form of nesting, however. Two communities may overlap in arbitrary ways as a result of objects filling roles in more than one community. This allows communities to be used to specify complex behaviour in a constraint-oriented style, with each community expressing a different set of requirements on the system being specified (components of which will fill roles in many of the communities being defined). In this way, the enterprise specification places obligations on the system being specified [4].

The main problem with handling the essentially social structures being supported by large open IT systems is that real organizations seldom adhere strictly to their own rules. They are often found in inconsistent states, so that some level of optimization and compromise is needed in interpreting such rules to manage automated parts of the system. This is a very different situation from that normally found in managing, say, computational interactions. Computational binding within the middleware can be controlled by a strict set of rules for matching interface types, formulated to ensure interaction is successful when using servers with more facilities than currently needed, or when system evolution is in progress. When considering an object taking part in an enterprise community, however, a different approach is needed. One way is to place more emphasis on the negotiation process, so that an object can agree to restrict its own behaviour to that which is required when joining the community. This leads to a distinction between the objects maximal possible behaviour and its currently agreed behaviour, called its social behaviour in [5].

Although it would be possible to express all aspects of enterprise behaviour from first principles by using suitable behavioural constraints on community membership, the practical application of these ideas depends on the creation of libraries of rules which are closer to and can be recognized as representing common business constructs. Examples are likely to be sets of general rules for describing, for example, authorization and delegation structures. More specific rules might cover particular accounting or resource allocation schemes.

How, then, are sets of rules and policies that are contain potential inconsistencies or conflicts to be interpreted? Guidance is still needed in particular circumstances on how an application should behave, or whether a particular middleware mechanism is to be included, and if so with what parameters or options. There needs to be some decision mechanism for resolving choices, even where there are contradictions. One possibility is to provide a system of priorities to select between opposing requirements, but this, on its own, is unsatisfactory for two reasons. First, a single system of priorities implies some degree of global view of the design, making federation more difficult to organize, and second the resulting behaviour is not

modified by the presence of the lower priority rule, so that the behaviour changes discontinuously as the priorities are varied.

An alternative approach is to express rules and policies in terms of a cost function [5], so that decisions are taken as a result of a notional optimization process. Strong rules correspond to sharply differentiated costs for the different courses of action, and weaker rules have smaller associated cost differentials. Each possible choice can be resolved by selecting the minimum cost path. This process depends on estimates of the expected behaviour of the environment in which the system is placed, and so is inherently adaptive. Various notions of obligation and prohibition can be modelled as changing the costs applied by the object accepting the responsibility.

Once a target series of actions has been identified, different strengths of infrastructure mechanism can be selected, based on the perceived costs of departing from the desired behaviour. On one hand, pessimistic mechanisms can check each interaction and block departure from the agreed sequence. This may be appropriate where the costs of violation are high, and the countermeasures cheap and localized. On the other hand, optimistic mechanisms may rely on objects satisfying the obligations they have undertaken and fall back on later corrective or punitive actions if there are exceptions. This may be the best solution if little is at risk and the checks to be applied are themselves distributed and costly.

These and other issues are the subject of active debate within the ODP community. The current working document for the Enterprise Viewpoint provides a basic set of definitions, augmented by specification structuring rules, correspondences with other viewpoints and a draft metamodel for the enterprise language. It is expected to be substantially complete before the end of 1999.

One of the requirements for a useful enterprise modelling notation is that it should be accessible to a wide and relevant community. This mitigates against exotic notations in favour of notations which are already familiar in at least the system analysis community. With this in mind, the Unified Modelling Language (UML) is being investigated by members of the ISO group to see to what extent it can be used as a basis for specifications in the ODP enterprise viewpoint. It is able to express simple conceptual structures, such as the relation of roles to communities, but cannot express objectives or policies in any general way. One possible direction is to work with a structure defined in UML and decorate it with a companion enterprise policy language, which would have the same sort of relation to the UML core as the existing UML Object Constraint Language does. Attempts have already been made to handle Quality of Service-related policies in this way [6].

3 NAMING AND FEDERATION

Probably the single most important step in federating independently originated systems is the establishment of rules for unifying the various namespaces on which the different systems depend. Because of the separate development histories of the different systems prior to federation, the interpretation and structure of their naming schemes may differ either in major ways, or in more subtle aspects of interpretation. Any use of a name which refers to something from across a federation boundary is likely to be problematic, because the rules in force when the object was first named and the rules applied where the name is being used will not, in general, be exactly

the same. Even if translation processes are agreed and put in place to handle gross differences, subtle variations are still likely to be present.

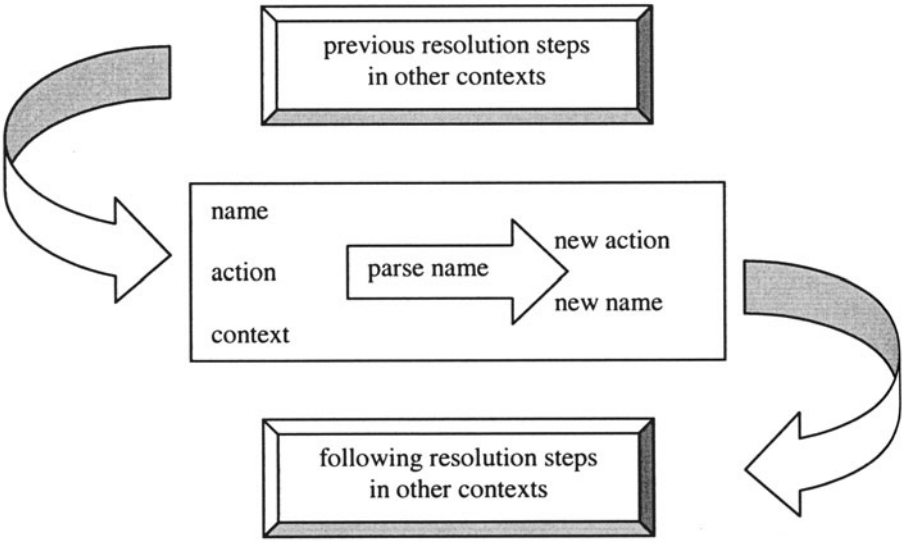

Figure 1. The name resolution process.

ODP takes as its starting point the recognition that all naming is context relative. Even if efforts are made to establish global naming authorities and unambiguous naming processes, there will still be variations of interpretation to be coped with. There will be different assumptions about the implicit properties of the things named, and the process of creating federations between systems with different technological bases will gradually erode uniformity. The ODP Naming Framework [7] acknowledges this process and makes mechanisms for coping with it explicit by associating a context with any action which involves naming.

When an action is performed which interprets a name, the name is processed in the appropriate context, qualified as necessary by the nature of the action being performed. This analysis may result in identification of the resources necessary to complete the action, but it will, in general, result in the identification of a further action to be performed. This will lead to the transfer of a modified or translated name into another context where the activity can be continued (see figure 1).

Distributed systems depend on the ability to transfer names in order to extend existing configurations and publish the availability of new services. When this transfer crosses the boundary between domains, there may be a need to transform the name being transferred so that it is still possible to interpret it in the receiving context. There are several ways identified in the standard of dividing the responsibility for this transformation between domains, but they all depend on some part of the system being aware that a name is being transferred. The introduction of

ubiquitous middleware is important here, as it implies that there is awareness at a suitable place in the system structure that names are being transferred. This is a consequence of the awareness within the system of interface types and, in particular, of the parameter types to be marshalled and unmarshalled, and suitable naming-related actions can be associated with this process.

Naming systems are particularly difficult to manage when they result from the integration of the individually developed naming schemes of separate organizations which have decided to cooperate or to merge some aspects of their activities. This is the essence of the federation problem. The naming framework provides a number of mechanisms for organizing the federation process. They are based on the creation of specific naming structures to localize the management of the federation process and decouple it from the normal evolution of the naming systems within the organizations. There are three main techniques:

1. creating an export context for the names of objects or services within your organization which are to be accessed from outside. The export context decouples internal and external names and can be used to control external visibility if name resolution requests are only accepted in the export context;
2. the two or more parties to a federation agreement creating a shared context in which each of their export contexts are named, thus providing a level of uniformity when exchanging names within the federation by ensuring that there is at least one context shared between the partners;
3. any organization may create an import context which maps names accessed via the federation context into convenient local forms, decoupling local usage from variations resulting from changes in the federation agreements.

The creation of a federation depends on the parties involved agreeing various obligations and responsibilities – essentially an enterprise specification. This includes the purpose of the federation, the communication mechanisms to be used, the form and content of the federation context and the names for suitable export contexts that the members undertake to support.

Once the necessary federation agreements have been put in place, and technical measures taken to support them, there remains the need to maintain the necessary information and to publish links to it. Part of this is a matter of supporting the naming process, but something more is needed to ensure consistent interpretation of the agreements. This semantic support is provided by the Type Repository.

4 TYPE REPOSITORY AND THE SHARING OF KNOWLEDGE

The common understanding of types is the basis for any form of communication and so the ability to organise such information is one of the essential planks for the support of system development. One of the first things to be done when establishing federation between systems is to establish correspondences between types.

Types are used in many aspects of system configuration. They are used to express requirements when trading, to check compatibility during binding, and to confirm consistency of implementation during compilation and component integration. The dependencies of system components on types are complex, and there are many different type systems which interrelate and overlap in a variety different ways. The approach taken in ISO [8] is, therefore, to provide a general mechanism for

describing the model that represents each type system, and to allow families of related type definitions to be described by higher-level models, or meta-models. In this way, support for a range of different techniques and notations can be provided, and there is a basis for relating the expressions of a single underlying type in a variety of languages.

The recursive use of meta-object definitions offers great expressive power, and there are few practical problems that require more than two or three levels of modelling to capture their type definitions.

The type repository provides a powerful link between activities taking part at different stages in the system's lifecycle. It acts as a common store for type information used to express requirements, outline designs, management constraints, policies and implementation details. It can also store the refinement relationships that link types in an abstract system view with more specific types used in a variety of implementations.

The packaging of the repository as a collection of objects accessible using the standard middleware also blurs the distinction between design time and run time. System components within the infrastructure can access type information deposited when the system was built, facilitating the provision of flexible channel components such as interceptors which convert from one data representation to another and simplifying interoperation between different implementation domains. Replacing a basic interface repository with a more general type repository simplifies the provision for dynamic invocation and makes possible the selection of marshalling and representation options at binding time. Techniques of this sort can simplify system evolution and the deployment of new services.

The ISO work on the definition of the Type Repository is now being based on the OMG definition of the Meta-Object Facility. The ISO standard defines the context for the work and the way in which it relates to the RM-ODP framework, but references the OMG document for the definition of the computational interfaces involved.

5 BUILDING CONFIGURATIONS AND BINDING OBJECTS

The RM-ODP introduces, as part of its computational language, the notion of binding. It goes on to qualify binding as being either implicit or explicit, and either compound or primitive. The first distinction deals with the visibility of the binding process in the computational virtual machine. The second centres on whether or not the binding that is produced is a first-class object which can participate in the behaviour of the system, interacting with other objects, so that it can be dynamically modified and controlled [9].

The new standard on Interface References and Binding [10] extends this model by giving a framework for the engineering support of the binding process. It introduces the concept of a binding factory, which is responsible for collecting the necessary resources and constructing the binding object requested. This factory negotiates with the infrastructure components in the systems which are supporting each of the objects to be bound, and performs checks to see that the interface types are compatible [11]. It then constructs a suitable channel between the endpoints, taking account of any quality of service constraints on the binding, by using the primitive bind operations of the technologies concerned. Finally, having checked that the

binding object is correctly initialized, the factory returns a reference to the binding's control interface to its client.

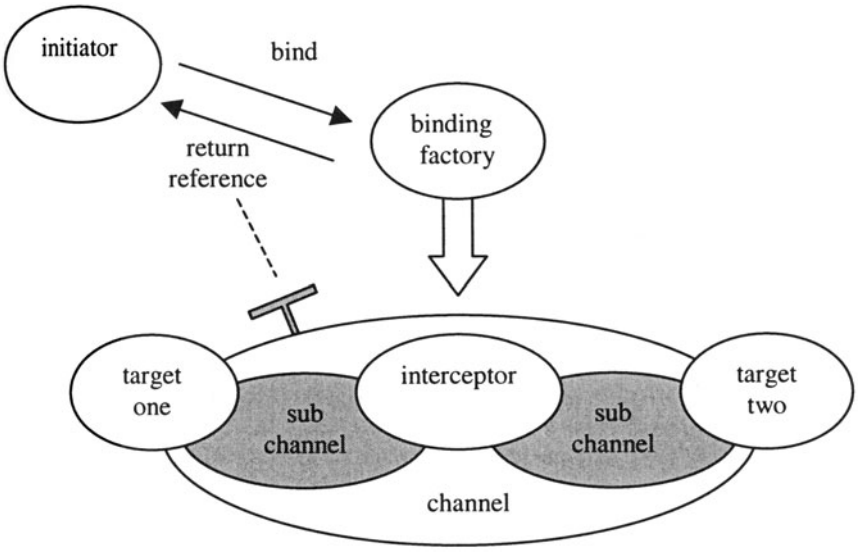

Figure 2. The binding process.

The factory is responsible for supporting policies concerned with achieving type compatibility and dealing with federation and the crossing of various kinds of domain boundary. This process may involve the allocation or creation of suitable interceptors along the communication paths; it can lead to a hierarchical process of channel creation. The establishment of path segments between interceptors is delegated to subsidiary factories, under the control of policies and sub-goals established by the top-level binder (see figure 2).

The need to manage the binding and to control resources does not stop when the binding is created. If the objects bound are mobile or persistent, there may be a need for significant reconstruction of the channels supporting the binding from time to time during its lifetime, particularly when objects migrate from one domain to another.

Perhaps the single most important requirement for an architecture to be considered open is that is should support a measure of transparency in its handling of names and references. If a system is to be federated with others of different ages and different supporting technologies, there will be a need for some adaptation measures at the boundary, such as protocol conversion or additional authorization procedures. These will often be performed by interceptors on the channel between systems in the two federated domains and will need to be set up when a binding across the federation boundary is created. This requires that suitable information about the nature of the required binding on both sides of the boundary should be provided with an address or reference presented with the binding request.

The operation of an interceptor will generally require information about the state of the dialogue to be provided by the initiator of an interaction. The initiator will

expect to provide information on its own state, but the interceptor must use addressing information to obtain some handle on the state in the other domain. Since this information may be quite bulky, an address or reference may include a pointer to its definition, rather than the information itself, but it still needs to be accessible at the time the binding is made.

To meet these requirements, ODP provides appropriate mechanisms. The step-by-step resolution of names has already been described. For interface references, the standard on binding provides another mechanism. These references are normally interpreted directly by the binder. However, an alternative format is defined which gives the identity of some supporting object, together with a transparent body of information to be passed to that object; it can then generate a suitable reference with which to construct the binding. Adding this additional indirection allows, depending on the nature of the supporting object accessed, the construction of various mechanisms for the dynamic creation of interceptors, performance of additional routing measures, firewall management and many other things. The form of interface reference defined for ODP is closely related to that used by OMG, but additional data tags are reserved for the transparent information to support federation, increasing the ability to deal with legacy systems and simplifying the integration of future generations of middleware.

One other important piece of technical work within ISO should also be mentioned as underpinning the binding process. This is the standard defining Protocol Support for Computational Interactions [12], which provides a vital link between the ODP architecture and the OMG CORBA definitions. It defines how the interactions in the abstract computational viewpoint of ODP are mapped directly onto the interoperable protocols (GIOP) which underpin CORBA, and so defines the basis for interworking between parts of ODP systems. The standard is written in terms of a general interworking framework, and so can be extended in future to define interworking not only for CORBA, but also between a variety of related mechanisms, as long as they are all able to support the same basic computational model.

6 FUTURE DIRECTIONS FOR TOOL BUILDING

Traditionally, there has been a fairly clear separation between design, system implementation and runtime operation and management. Design tools used to develop high-level object designs may have included code generation facilities for the creation of implementation skeletons, but the designs were stored in a format specific to each tool. They were not available for use in later stages of the implementation process, and quite separate from the runtime environment.

Significant benefits can be achieved by strengthening the linkages in the tool chain, so that automatic generation and checking processes can be steered by higher level information. One example of this developed at Kent is the use of a variety of information to construct performance models for new applications early in the design cycle. This project, called Permabase, was carried out jointly with British Telecom, and took UML designs, together with configuration information on the target platforms on which the application was to be run, and made performance predictions for a variety of expected workloads [13]. The system successfully predicted performance if the designs were reasonably complete, but would need access to other sources of information derived from the orgaization's policies and prior experience

to make predictions really early in the lifecycle, when designers have articulated only a hazy view of the behaviour required. This illustrates the general need to have access to as wide a variety of information as possible in order to improve the performance of individual tools.

Another example of the use of high level information is in the implementation of security policies. Policies may be stated in organizational terms, categorizing information and use by, for example, organizational roles or departmental functions. The interpretation of these policies by information providers requires them to be given appropriate information about their users which needs to be supported by the middleware if it is to play a significant role in authentication. Combination of security policy and organizational information would allow much finer-grain checking of specific network paths than would be possible with manual configuration, making internal hierarchies of fine-grain firewalls practical.

To achieve a higher level of integration, there needs to be a change of emphasis, so that the design and configuration information for a system is seen as a resource in its own right, which has many users, including, but not limited to, the tools which traditionally manipulate it. Tools need to be modified in three ways:

1. they need to be repository-based, so that the information they manipulate can be held in open formats and accessed concurrently by a wider range of components;
2. they need to be able to respond to events signalled by the repository, so that they can take account of changes as they occur, allowing multiple tools to communicate via the information they share. This is particularly important when inconsistencies between different views are detected, or if, for some other reason, there is a need to highlight to the user contributing elements in views managed by different tools. In the performance tool mentioned above, for example, one would wish to highlight performance-critical components directly in the existing user interface to the designer's view;
3. they need to respond to requests from other tools to perform their checking and validation actions, to avoid duplication of function in different tools. Thus if a resource control tool attempted to modify a configuration, it might request an enterprise description tool to check a wider range of policies, flagging, for example, that the proposed change should be rejected as conflicting with, say, a resilience policy.

Note, however, that this does not require a monolithic software development environment. What is proposed is a federation of tools and repositories in the style discussed above for distributed systems in general. In order to achieve the necessary management flexibility, it would be necessary for the different tool domains to retain significant independence. Indeed, one of the weaknesses of current repository designs is the lack of a sufficiently powerful federated versioning model to support the overlapping requirements and activities found in large systems.

Consider, for example, the problems faced by two organizations with an existing federation agreement when they identify a need to add additional audit information to their interactions. Suppose that, for local reasons, one organization needs to make corresponding internal changes urgently, but the other does not. Revisions are made to the shared information model to give a single definition of the new information, and these new items are referenced in the federation agreement. It may be that, at this stage, some interactions with an existing policy on privacy are detected and need to

be resolved. The first organization then starts development and corresponding types, such as the new IDL definitions, appear in its type repository and begin to be used internally.

If a binding is now requested for this interface on a path between the two organizations, the binding factory may detect the need for an interceptor at the domain boundary. This interceptor will be instantiated dynamically, and will be configured using type repository information so as to convert between new and old style interactions, and add information from a suitable policy on defaults where necessary. The creation of the interceptor will commit resources and this may feed back as a need for action seen in some capacity-planning tool.

While this thread of activity is going on, other developments will also be under way. They will each need to pass through a number of approval steps before becoming operational, and so support for separate testing and operational versions of components and their specifications will be needed. However, the point at which plans become visible will vary from tool to tool. It would be desirable, for example, for the capacity-planning tool to have some visibility of the changed resource requirements of the application before the changes come into operation, but an operational binding factory should clearly not normally take any note of development versions. Whether parallel development activities should depend on each other's predicted products will need to be decided on a case by case basis.

A new application development is likely to start, at least, with a phase of top-down activity. An enterprise model will be constructed early, identifying some policies from known requirements and inheriting others from established norms for the organization. The enterprise model will contain enough detail to express key use-cases and may be able to generate, in skeleton form, some parts of the computational design. As the design proceeds, periodic checks will be made to see that policies are not violated (although the design may pass through inconsistent states as a result of restructuring during development). More importantly, checks made during testing should help to detect unintended violation of policy that could occur while correcting errors. Finally, the policies can be applied to select options and assist in configuration when the application is deployed. Middleware transparencies and Quality of Service targets can be derived in part from the enterprise policies, together with analysis of the application dependencies and infrastructure configuration.

What of changes in policy when a system is already operational? One requirement is to be able to assess the consequences of changes in policy on the application. Some changes may be directly applicable to the running system, via communication of the changes to appropriate management objects. A sufficiently flexible middleware or network management system could respond directly to a change in policy, but should only do so in response to a specific performative act, allowing assessment of policy change without immediate consequences. Other changes might alter the way that processes such as binding are performed, and would take effect progressively as new activities start. Yet others might require development, modifying parts of the design to implement the policy. Using the tools to check the scope of changes required to implement the proposed policy could provide important information on the economics of the proposed change.

If a decision is made to implement the change, a plan for the evolution of the system will be needed, and this, too may be simplified by the bringing together of

information from different tools and management domains to identify short, feasible transition sequences.

How, then, do the current ODP activities fit into this vision of tool integration? They provide guidance for organizing the process and some of the key components needed to bring it about. Firstly, the whole idea of federating a wide range of tools is only really plausible if there is a ubiquitous middleware to enable open communication between the components. Given this base, we can expect:

1. the enterprise language work to provide a framework for capturing information on organizational structures and policies to constrain and guide all aspects of the system's lifecycle;
2. the naming framework to be used to identify the different contexts and actions to be taken when changing context;
3. the type repository to provide one of the key integrating mechanisms by allowing the sharing of a wide range of specification information between many kinds of tools and runtime components;
4. the binding model and the transparent mechanisms provided within interface references to give late binding and flexibility in resource optimization and the dynamic interpretation of policies.

7 CONCLUSIONS

The recent work on ODP standards provides a powerful set of additional models and frameworks to support the creation of large distributed systems. They give the opportunity for information specified in a number of viewpoints to be combined by suitable tools to increase the level of automation in system implementation, configuration and management.

Use of repository technology to link the various steps in the tool chain with the run-time checking and interpretation of policies within the middleware itself should lead to more robust and flexible systems, capable of evolving to meet the changing requirements of large scale, federated distributed systems.

8 ACKNOWLEDGEMENT

The ideas in this paper are derived in large part from discussions in the ISO ODP group and the author acknowledges the contribution that these stimulating discussions have made. However, the responsibility for interpretations and predictions of likely future directions remains with the author.

References

[1] ISO/IEC IS 10746-2, *Open Distributed Processing Reference Model – Part 2: Foundations*, January 1995.

[2] ISO/IEC IS 10746-3, *Open Distributed Processing Reference Model – Part 3: Architecture*, January 1995.

[3] ISO/IEC WD 15414, *Open Distributed Processing – Enterprise Viewpoint*, January 1999.

[4] TYNDALE-BISCOE S. AND WOOD B., Machine responsibility – How to deal with it, *Proc. 1st International Workshop on Enterprise Distributed Object Computing (EDOC'97)*, Gold Coast, Australia, October 1997.

[5] LINNINGTON P., MILOSEVIC Z. AND RAYMOND K., Policies in Communities: Extending the ODP Enterprise Viewpoint, *Proc. 2nd International Workshop on Enterprise Distributed Object Computing (EDOC'98)*, San Diego, USA, November 1998.

[6] AAGEDAL J. AND MILOSEVIC Z., Enterprise Modelling and QoS for Command and Control Systems, *Proc. 2nd International Workshop on Enterprise Distributed Object Computing (EDOC'98)*, San Diego, USA, November 1998.

[7] ISO/IEC DIS 14771, *Open Distributed Processing – Naming Framework*, July 1998.

[8] ISO/IEC FCD 14769, *Open Distributed Processing – Type Repository Function*, January 1999

[9] BLAIR G. AND STEFANI J-B., *Open Distributed Processing and Multimedia*, Addison Wesley, 1998.

[10] ISO/IEC FDIS 14753, *Open Distributed Processing – Interface References and Binding*, September 1998.

[11] KUTVONEN L., Sovereign Systems and Dynamic Federations, *Proc. 2nd International Working Conference on Distributed Applications and Interoperable Systems (DAIS99)*, Helsinki, Finland 1999.

[12] ISO/IEC DIS 14752, *Open Distributed Processing – Protocol Support for Computational Interactions*, January 1999.

[13] WATERS A. G., LINNINGTON P., AKEHURST D. AND SYMES A., Communications software performance prediction, *13th UK Workshop on Performance Engineering of Computer and Telecommunication Systems*, Ilkley, West Yorkshire, July 1997. BCS Performance Engineering Specialist Group.

Biography

Peter Linington is Professor of Computer Communication at the University of Kent at Canterbury in the UK. His research interests include distributed processing architectures, multimedia systems, and the monitoring and performance of broadband networks. He has been involved in standardization since 1978, and has worked on ODP since the activity started in ISO. Before moving to Kent he was head of the UK's Joint Network Team, responsible for the development of the JANET network.