\documentclass[12pt]{report}
\pagestyle{plain}

\chapter{Interoperable Systems: an introduction}

\section*{Anthony Finkelstein}

\section{Scope}

This short chapter introduces interoperable systems and attempts to distinguish the principal research strands in this area. It is not intended as a review. Significant review material is integrated with each of the succeeding chapters. It is rather intended to whet the appetite for what follows and to provide some initial conceptual orientation.

This book concerns the architecture, modelling and management of interoperable computing systems. Our collective research agenda addresses all aspects of interoperable systems development, including the business and industry requirements and environments for distributed information services.

Our definition of interoperability is a broad one (it can be usefully contrasted with the rather narrow definition implicit in Wegner, 1996). Interoperable systems are systems composed from *autonomous*, locally managed, *heterogeneous* components, which are required to *cooperate* to provide complex *services*. Such systems are generally *distributed* and have significant non-functional constraints on their operation. The systems are *open* and are subject to continuing change. Interoperability is that property of components and of the means by which the components are composed which ensures that they jointly meet the service requirements placed on them. In other words interoperability is the ability of components, and groups of components, to interact effectively to achieve shared goals.

It is worthwhile examining each of the elements of this definition separately.

Autonomy is the most important characteristic of an interoperable system. It means, in essence, that each component of such a system can operate independently of the other components in its environment. It implies a degree of context independence and of local management. An autonomous component can be thought of as analogous to what is coming to be known as a computational agent.

Such autonomous components may be implemented or realised in different ways. They may be based on different programming technologies or embed a different model of their environment. They may achieve similar goals in very different ways. They are, in short, heterogeneous.

Clearly a collection of autonomous and heterogeneous components do not constitute an interoperable system. Interoperability arises from the need of the autonomous components to cooperate that is jointly contribute to the provision of a service. The essential challenge of providing support for the construction of interoperable systems is to establish the best balance between the compatibility required for cooperation and the concommitant loss of autonomy and

heterogeneity.

A service is an activity or set of activities performed by a component or set of cooperating components on behalf of a client, which may itself be a component, set of components or external agent. What distinguishes a service from the conventional conception of a function is its granularity - it tends to be a relatively complex composite - its longevity - it tends to be relatively shortlived, services are frequently changed and reconfigured - its independence - a service has an independent definition which may not be in the control of any of the agents contributing to it. Ideally the implementation of a service is transparent to the clients of that service. For a very interesting discussion of services see Cohen (1995).

Generally clients have strong requirements on the quality of service. This has two aspects: the service must accurately implement the service definition; the service must meet so-called "non-functional" requirements such as performance, security, dependability and so on. What make the non-functional requirements of particular concern is that they cannot be simply determined from the behaviour of the individual components but instead from the composition of the components and the means by which they cooperate.

Our definition of an interoperable system subsumes logical distribution. Physical distribution is however not a defining characteristic of an interoperable system but most interoperable systems of the type discussed below are physically distributed. This means that the support for cooperation between components must be appropriate to distribution with all that this implies. It also means that heterogeneity of components may be accompanied by heterogeneity of the platforms on which those components reside

Openness is a term used in a variety of senses. By openness in this context we mean that components, and of course services, may be added or removed from the system at any stage. The external agents who do this are themselves autonomous and there are no guarantees that they will add these components in a safe or consistent manner. An underlying assumption is that at least some of these agents may be competitors who wish to offer the same services with "improved" quality of service.

Interoperability is the ability of components to cooperate to achieve shared goals or to participate in services that require the components to interact. In such a setting an external agent can set service requirements and can establish whether these service requirements will be met

Motivation

Given the nature of the definition above it may seem unncessary to provide an extensive motivation for the study of interoperable systems. However, because interoperable systems have rarely been studied as a coherent class of system it is worthwhile to review the main reasons why they are worthy of attention. This motivation discusses why interoperable systems are important from an industrial/business perspective and complements this by considering some example domains.

It is worth dividing the industrial/business concerns with interoperable systems into two further aspects loosely termed push and pull. On the push side businesses are finding

themselves with systems which must interoperate. They have legacy software which incorporates important business information or controls key devices and is therefore business critical. These pieces of software must work together with other pieces of more recent software, which may be purchased off-the-shelf. The legacy software is autonomous, in the sense of being designed and built for stand-alone use. It is also heterogeneous being based on legacy software technologies and platforms. Interoperability provides a way to work with the mess that constitutes many organisations software base. Because most industrial/business concerns are not as coherent or organised as software system engineers would demand they be 'push-interoperability' is not just an issue of legacy, parts of large decentralised organisations are continually acquiring software and subsequently discovering that they would like these to cooperate.

'Pull interoperability' is a feature of the highly dynamic environment in which most industrial/business concerns find themselves. Commercial advantage accrues to the organisation which can most rapidly innovate new services or respond to services offered by their competitors. Most such services, think for example of airline traveller incentive programmes, require a new set of software services to be built across existing components.

Added to these is the fact that there are emerging market opportunities in existing interoperable settings, for example by providing domain interfaces (aka vertical facilities) within CORBA or value added services over the Internet.

To appreciate these concerns it is worthwhile briefly considering some example domains where interoperability is a critical issue.

Mobile Computing — Conventional assumptions about the future of computing have concentrated on workstations with extensive memory, sophisticated interfaces and high bandwidth communication. These assumptions have been subverted by the trend towards mobile devices such as pagers, active badges, pocket computers, hand-held scanners, personal entertainment systems and so on. These devices use cellular radio, infrared or other wireless communication. There is a major commercial interest in developing services which incorporate these devices - mobile Internet access, sports results delivery, security tagging and tracking - the list is almost endless. Mobility implies distribution and, commonly, fluid connection patterns. Components are added and removed with considerable frequency. The services, which change rapidly, depend on a technologically heterogeneous infrastructure delivered and controlled by many parties, device vendors, communications providers, service suppliers. The limitations of the devices and communications place particular emphasis on global service requirements such as fault tolerance.

Electronic Commerce — Anybody who has used the Internet, and particularly those who have ordered books, T-shirts and CD-ROMs through WWW, will be aware of the growth of electronic commerce - markets mediated by computing and communications technology. Advertising, search and evaluation of competing products, bidding, bid assessment, contracting, payment and even delivery, in the case of digital services such as video, music and software, are being undertaken within networks of globally distributed computing resources. The ability to design, deliver and exploit such electronic commerce services rapidly confers significant business advantage. The growth of services such as independent brokering forces the pace of interoperability. The services present important non-functional requirements, of which

notable examples are security and confidentiality.

Software Engineering — The infrastructure required to support software engineering is itself a particularly interesting example of the class of system we are discussing. Software development is often carried out in a distributed setting with many different, partially or wholly autonomous, individuals, groups and organisations participating. The information that is produced and deployed is heterogeneous. The need to engage in cooperative work and to integrate elements developed in this setting present service requirements for an appropriate development environment.

Telecommunications — The change in the commercial and regulatory position of telecommunications companies set alongside technological change has raised a new set of challenges. Telecommunications providers are increasingly concerned with developing user-oriented services, as distinct from simply selling telephone calls. These services, features in telecommunications terminology, must operate in a heterogeneous setting with equipment from a variety of suppliers and must be compatible with services provided by other vendors and with other services that they are providing. Elements of the services are to be provided by legacy systems whose behaviour is ill-understood and for which change and evolution are particularly problematic. As in the focus applications discussed above, there are critical non-functional requirements constraining service delivery, in this case primarily reliability and availability.

Research Contributions

From a research standpoint interoperability brings together many of the most critical issues in computing research. Autonomy, heterogeneity, distribution and openness manifest themselves in many different forms and cut across many conventional divisions within computing. Research on interoperability is about finding working solutions to the challenges of interoperability without excessive knock on consequences.

Autonomy embraces many of the issues presented by component technologies, the challenges are those of developing a sound encapsulation scheme and a means of realising components which do not have side effects on each other.

Heterogeneity involves being able to map between languages, either directly or across some wrapper or intermediate interface layer. The research challenge lies in identifying the mappings and designing wrappers and interface layers. Identifying the mappings may range from the relatively straightforward relationships between certain programming languages to the complexities of mapping between different data models and query schemes implemented in different databases.

Distribution is still a significant issue. Designing distributed systems at a realistic scale and with the required levels of dependability lies at the state of the art of industrial practice. In research terms being able to manage such systems is of continuing interest.

Openness remains the most slippery of the issues in the sense that it presents such a wide range of related challenges. The most important of these is the provision of mechanisms which allow components to be added and removed from running systems. With such basic mechanisms in place the issue of safe system evolution becomes critical.

Threading through, and uniting all these research issues is a single common problem - composition. How, when you bring components together can you gain some assurance of their joint behaviour. More particularly, how can you gain some assurance that properties of components, or sets of components, that had previously held, will continue to hold when further components are added. Without such an assurance, in an open setting services may interact with each other in unanticipated and undesireable ways - feature interaction in telecommunication terms. This is a foundational concern within computing and seems unlikely to yield to an immediate solution.

Figure 1 below shows the principal contributory components of the research on interoperable systems and the most important relations between these components.

We distinguish two basic threads: a systems technology thread, focusing on the software and systems required to support interoperable systems; a software engineering thread, focusing on the development and evolution of interoperable systems. Intersecting with these we can see three major research areas: modelling and specification; distributed systems; systems architecture. In each of the areas the main elements of the research are set out: within the software engineering thread these are development activities; in the systems technology thread relations are the layers of system services. The arrows mark the important relations, which is where the immediate challenges lie: relating requirements to application architecture; mapping applications to interoperation infrastructure; relating operating system services to system architecture; relating system management and evolution to the capabilities of the infrastructure; and so on.

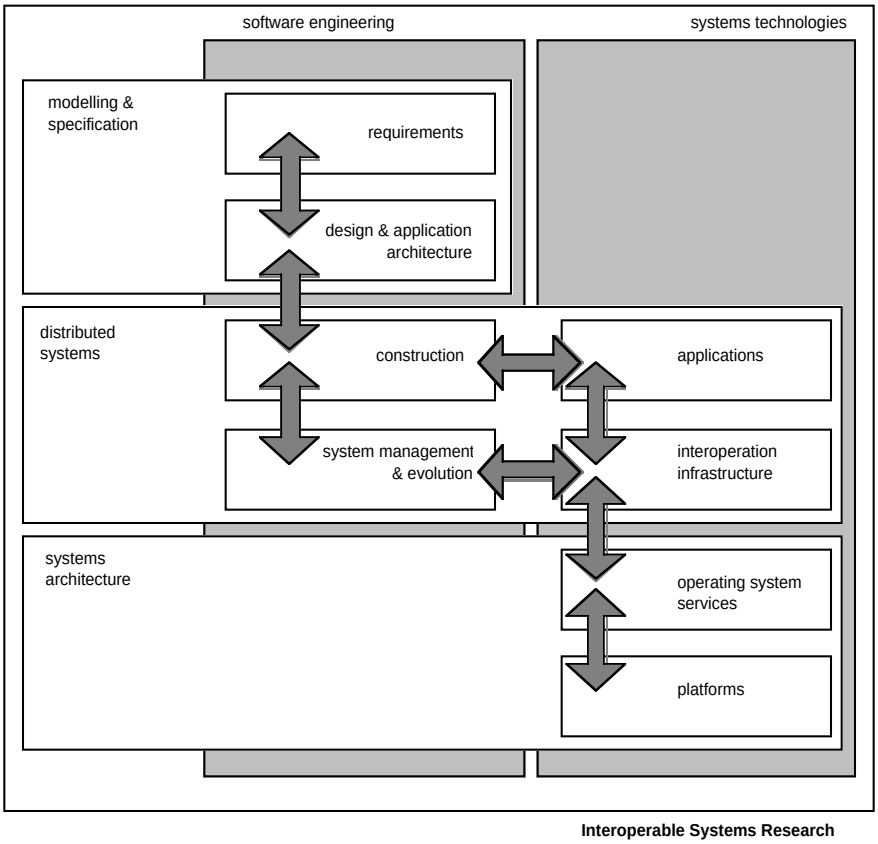

Figure 1.

Inevitably not all aspects of interoperability are covered in the book, but a considerable amount of the most important ground is covered. Interoperation infrastructures are extensively discussed in our account of emerging industry standard middleware (CORBA and DCOM) and how such infarstructures can be augmented. System management is discussed in the context of interoperation infrastructures. Composition is treated both formally, in an analysis of compatibility, and from a software architectural standpoint. The book covers in some detail interoperability in the context of data management. This forms part of an attempt, regretably only partially successful, to present the issues of database interoperability and distributed object interoperability within a common framework. From a practical point of view databases consitute both a common class of legacy component and data management a common class of service required in an interoperable setting. Indeed, it can be argued that because of issues such as charging for, auditing the use of, and brokering interoperable services the ability to achieve database interoperability at all levels but most particularly the semantic level is a prerequisite for the provision of generally interoperable systems.

Future

The work reported in this book constitutes the starting rather than the stopping point of work on interoperable systems. We have reached a point where, perhaps, we have a more accurate understanding of the nature and scale of the challenges. In the terms of \cite{brooks87} we are able to distinguish the "essence" of interoperability, discussed above, from the "accidents" such as conflicting standards, particular wrapper technologies and so on. Just as in standard

systemdesign we have moved to a situation in which a 'normal', 'common or garden' application is distributed and concurrent, so we predict a situation in which all systems will be required to be interoperable. The problems of achieving interoperability will not be the province of, for instance, specialists in legacy systems integration or telecommuncations system developers, but will be those shared by all programmers.

Acknowldegements

This work was supported by the EC Europe-Australia Cooperation Project ISI. I would like to acknowledge the contribution of Jeff Kramer and Jeff Magee with whom I have had a number of most interesting discussions on this topic.

Bibliography

\bibliographystyle{plain}
\bibliography{acwfbib}
\end{document}