European Cooperation
in Science and Technology
- COST -

Secretariat

Brussels, 4 July 2012

COST 4132/12

MEMORANDUM OF UNDERSTANDING

Subject :	Memorandum of Understanding for the implementation of a European Concerted Research Action designated as COST Action IC1201: Behavioural Types for Reliable Large-Scale Software Systems (BETTY)

Delegations will find attached the Memorandum of Understanding for COST Action as approved by the COST Committee of Senior Officials (CSO) at its 185th meeting on 6 June 2012.

MEMORANDUM OF UNDERSTANDING
For the implementation of a European Concerted Research Action designated as

COST Action IC1201 BEHAVIOURAL TYPES FOR RELIABLE LARGE-SCALE SOFTWARE SYSTEMS (BETTY)

The Parties to this Memorandum of Understanding, declaring their common intention to participate in the concerted Action referred to above and described in the technical Annex to the Memorandum, have reached the following understanding:

1. The Action will be carried out in accordance with the provisions of document COST 4154/11 "Rules and Procedures for Implementing COST Actions", or in any new document amending or replacing it, the contents of which the Parties are fully aware of.

2. The main objective of this Action is to develop improved programming languages and tools, based on behavioural type theory, for the implementation of reliable large-scale distributed software systems.

3. The economic dimension of the activities carried out under the Action has been estimated, on the basis of information available during the planning of the Action, at EUR 48 million in 2012 prices.

4. The Memorandum of Understanding will take effect on being accepted by at least five Parties.

5. The Memorandum of Understanding will remain in force for a period of 4 years, calculated from the date of the first meeting of the Management Committee, unless the duration of the Action is modified according to the provisions of Chapter V of the document referred to in Point 1 above.

____

A. ABSTRACT AND KEYWORDS

Modern society is increasingly dependent on large-scale software systems that are distributed, collaborative and communication-centred. Correctness and reliability of such systems depend on compatibility between components and services that are newly developed or may already exist. The consequences of failure are severe, including security breaches and unavailability of essential services. Current software development technology is not well suited to producing these large-scale systems, because of the lack of high-level structuring abstractions for complex communication behaviour.

This Action will use behavioural type theory as the basis for new foundations, programming languages, and software development methods for communication-intensive distributed systems. Behavioural type theory encompasses concepts such as interfaces, communication protocols, contracts, and choreography. As a unifying structural principle it will transform the theory and practice of distributed software development.

The significance of behavioural types has been recognised world-wide during the last five years. European researchers are internationally leading. There is an urgent need for European co-ordination to avoid duplication of effort, facilitate interactions among research groups, and ensure that the field proceeds efficiently from academic research to industrial practice. This Action will provide the co-ordination layer and leverage the efforts of European researchers, to increase the competitiveness of the European software industry.

Keywords: distributed software infrastructure, software development methodology, foundations of programming languages, service-oriented computing, behavioural types

B. BACKGROUND

B.1 General background

Modern software systems operate on a large scale. They are distributed, collaborative and communication-centred, and are an essential part of the technological infrastructure of society. Many critical services are offered via the Internet and depend on other network technologies such as Clouds. These services include commercial operations such as banking, e-commerce, social networking and document sharing applications, as well as public-sector systems such as infrastructures for local, national and trans-national e-governments, e-healthcare and e-science. They are long-lived and run continuously, constantly collecting information and dynamically acquiring new functionalities, while network-aware devices that access these services, including mobile phones and tablets, adapt dynamically to their environment.

Recent years have seen the development of a new paradigm for the development of distributed software systems, based on the internalisation of the concept of a distributed service, often called service-oriented computing. In this paradigm, the computational behaviour of a system is organised into a collection of services, which run concurrently and interact with each other, instead of being organised into a collection of functions which run sequentially. This collection of services implements another service. For example, a login service for a popular web portal is usually implemented by multiple distributed components realising more specialised functions, such as an external identity provider, a browser, an internal identity registry, and an internal authentication service.

How can improper, undesired behaviour be prevented in complex and socially critical distributed applications? Protocol incompatibilities, malicious resource usage, security breaches and deadlock/livelock can have wide-ranging consequences, from temporary service outage to information leakage to exploitation of security vulnerability by criminal organisations, affecting a large number of users.

In traditional sequential programming, the systematic and rigorous application of data types and type checking has yielded huge improvements in software reliability and programmers' productivity. The pinnacles, so far, of type theory in mainstream programming languages are the sophisticated generic type systems of Java and C#, where programming abstractions are built in at a fundamental level. Data types abstract and specify the behaviour of permitted *operations* on data. Even if types are not used for static checking, they often play a fundamental role in runtime checking (as in Python, a language widely used for server-side applications). Because data types offer basic abstraction of programs' behaviour, cutting edge logical verification tools such as Java Modelling Language (based on Java) and Spec# (based on C#) articulate their logical assertions for programs by building on the type abstractions of the respective programming languages.

Thus data types offer an effective basis of programming and verification for sequential data processing. For building distributed, service-oriented computing software, a corresponding structural view requires *behavioural types*, which specify the permitted *interaction* within a distributed system. The behavioural type of a service specifies its expected patterns of interaction using a simple, expressive type language, in a way that can be used to determine automatically whether a service interacts correctly with other applications. At present, although there is a significant body of theoretical work on behavioural types and a growing collection of prototype programming language designs, it is not yet possible for industrial software developers to exploit behavioural types in the design, analysis and implementation of service-oriented computing systems.

The goal of this Action is to transform the theory and practice of service-oriented software development by the systematic application and unification of behavioural type systems. The result will be new foundations for service-oriented systems, new programming languages whose designs are informed by behavioural types, new program analysis tools based on behavioural types, and new software development methodologies based on these foundations, languages and tools.

In order to achieve this goal, significant research challenges need be tackled.

- How can mathematical theories of behavioural types be unified, capturing different aspects of software behaviour at levels ranging from bare protocols to more refined properties such as security policies, and together offering a consistent, comprehensive foundation for the large-scale distributed infrastructure which constitutes global service-oriented computing?
- How can behavioural types be integrated into fully-fledged language designs, compilers and tools, both as new production-level programming languages and as enhancements of widely used programming languages, so that software developers can effectively structure their code around behavioural types?
- How can behavioural types be used in practice as a basis for specifying and verifying the correctness of software components in large-scale, long-lived distributed systems, both at development time and at runtime, so that effective validation of components and services is possible?

Recently there has been a rapid growth of activity in the field, and an accelerated expansion of the research community. There is a danger that a proliferation of alternative and incompatible approaches will lead to duplication of effort, hindering the transition from foundational theories and prototype programming languages to industrial-strength tools and methodologies. There is therefore a need for direction and co-ordination. On the basis of a large number of research groups and many national research projects, these activities will reach a tipping point towards their practical world-wide adoption by having a supporting structure within which to organise the community, with funding for strategic meetings and scientific exchanges. A COST Action is an excellent mechanism for this purpose.

B.2 Current state of knowledge

Current best-practice in industry

Some of the better known existing large-scale distributed infrastructures include the backends of Amazon, Google and Microsoft Windows Live. Widely-practised methods for building distributed services in such large-scale infrastructures include:

- **Remote procedural call (RPC)**, or its object-oriented variant, remote method invocation (RMI). For example, a major part of Google's backend is implemented in C++ with RPC, together with the serialisation formats of Google Protocol Buffer.
- **Web service technologies through HTTP/SOAP** (Hypertext Transfer Protocol/Simple Object Access Protocol), which often use WSDL (Web Service Decription Language, a simple interface language). While the shape of interactions is essentially the same as RPC, all resources are accessible through HTTP, a single application-level protocol.
- **Messaging**, which often uses messaging brokers, centring on a publish-subscribe model. This is based on asynchronous message passing and is especially useful when rapid and reliable dissemination of data is necessary.

Another traditional method for integrating distributed components is to have implicit data flow through shared state. However, the explicit communication-based frameworks listed above have the merit that the infrastructure is clearly divided into distributed components with explicit interfaces; this makes it easier to export services, change service interfaces, and transplant a system into a new computing environment such as clouds. For example, during the last decade Amazon converted all of the software components in its infrastructure so that they are based on explicit communication interfaces centring on RPC/RMI, HTTP/SOAP and messaging. The framework Amazon has achieved is commercially known as SOA (Service Oriented Architecture).

RPC, RMI and HTTP are all based on the request-reply interaction pattern, which is a distributed version of a function call to data structures and objects. The restriction to the request-reply pattern causes several problems when, like Amazon, one imposes strict communication interfaces among distributed components. These problems include the lack of description or abstraction for overall interaction patterns (e.g. a sequence of interactions constituting a workflow is divided into numerous request-reply interactions) and latency due to the waiting time needed in request-reply. They are closely related to the lack of uniform programming abstractions capturing, among others, the above three forms of communication interfaces and beyond, and the lack of clean programming language support for distributed computing.

Production-level programming languages for building distributed services

In major programming languages usable for production-level application development, such as Java, C#, Python, Javascript and C++, there is no language-level support for distributed communication except as libraries (APIs, application programming interfaces). An exception is remote method invocation (RMI), which is part of the language specification in Java and C#, but this is one of the lowest-level communication primitives. This lack of language-level support means that communication patterns are not easily visible to the compiler or to program analysis tools. For example, implementing a login service requires a complex sequence of RMIs on constituent services. If this sequence is represented *only* as a series of API calls, with no explicitly-defined high-level structure, then programming errors are easy to make and difficult to detect.

The only relatively widely-used production-level programming language which centres on message passing is Erlang, which is roughly based on Hewitt's actor model. The main programming-level abstraction is the queue, which concerns only messages to a single actor. Another widely-used system, especially in scientific computing, is MPI (Message-Passing Interface). MPI is provided as an API (procedure library) for standard languages such as C and Fortran, and so it suffers from the lack of high-level abstractions mentioned above in relation to RMI. Furthermore, the MPI libraries are tailored for typical computational patterns in scientific number-crunching and are unsuitable for general-purpose distributed programming. Finally, although it is a research programming language, Occam-Pi is highly efficient but is again essentially based on synchronous handshake communication.

Recently Google has been developing two general-purpose programming languages based on its analysis of internal large-scale distributed software development. The Go programming language (http://golang.org/) is a system-level, garbage-collected, type-safe programming language which incorporates handshake communication. It is intended as an advance from the C programming language. Dart (http://www.dartlang.org/) is a class-based object-oriented language with pluggable static typing and with actor-like queue-based message passing communication. It is intended as an advance from Javascript, and is being developed as part of a large programming environment project. In both languages, communication is provided by language primitives, not by libraries. The underlying communication model used by Dart, based on asynchronous communication, is close to the lower-level communication model adopted in many languages based on behavioural types (see below).

Go and Dart show that in the context of the development of large-scale distributed applications, engineers have started to realise the need to treat communication seriously. But even in these latest programming languages, there is no language-level abstraction or specification mechanism for communication patterns beyond handshake or request-reply.

Origins of behavioural types

Types articulate computation in a given paradigm. The traditional notion of types offers abstractions for data, objects and operations on them, which are the key building blocks of computation in traditional programming languages. The basic form of behavioural types, specifying interaction patterns in distributed services, articulates the ways in which interactions are programmed and performed. From this basis, it is possible to specify more refined behavioural types and logical assertions. Thus behavioural types are not only a basis for verification, but also a way to organise computation in service-oriented, distributed computing systems. By centring on interaction, behavioural types are often automatically *compositional*. The strategy is to first verify whether or not the interactions of each service satisfy their behavioural types. After that, the behavioural type of a composite service, built from interactions between components, can be verified without re-examining the components.

Type theories for distributed computing systems are a new field largely initiated by the study of behavioural types. They are built on the formal foundations provided by process calculi, such as ACP (Algebra of Communicating Processes, CSP (Communicating Sequential Processes, CCS (Calculus of Communicating Systems), pi-calculus, and higher-order pi-calculus. Process calculi offer a rigorous mathematical basis upon which communication behaviours of software systems are represented, reasoned about and analysed. Combined with other formalisms such as Petri Nets, they offer a general framework in which to mathematically capture diverse forms of interaction, rigorously and in a distilled form, including synchronous, asynchronous, untimed, timed, and others.

For theories of behavioural types, process calculi with channel passing capability, starting from pi-calculus and higher-order pi-calculus, play a central role. They offer a rich basis for behavioural types because their tiny syntax (a fully expressive asynchronous version of the pi-calculus can be defined by a standard Backus-Naur Form grammar in six lines) can nevertheless express fully the whole range of interactional behaviours realisable by programming languages, thus serving as a rich basis for investigation.

However, calculi and programming languages differ in that the latter also need to offer programming abstractions (structuring principles for software) and efficient executions, among other pragmatic concerns. Calculi offer a basis upon which these ideas can be formally expressed and examined, but it is also necessary to build a comprehensive programming framework which offers effective structuring principles and other foundations for building distributed, service-oriented computation.

Research into behavioural types and the associated programming primitives has been addressing these concerns.

State-of-the-art in behavioural types

Representative technical approaches to behavioural types include:

- *Types for sessions* (session types), which are based on the idea that interactions among distributed components and services can naturally be divided into multiple conversations (sessions), each with a specific protocol. The division into sessions enables tractable type-abstraction, making it easy to specify protocols and validate programs.
- *Contracts for interactions*, which give fine-grained specifications stipulating reciprocal responsibilities among interacting parties, often using labelled transition relations. Contracts can be combined using various algebraic operators to obtain larger contracts.
- *Behavioural security specifications*, where security properties are specified as behavioural types, often elaborating simpler notions of behavioural types. This enables an explicit specification of information flow as reciprocal responsibilities among communicating parties in a conversation, offering a uniform framework to specify distinct security properties such as secure information flow and access control, centring on types for interaction.

There are diverse sub-approaches in each domain: for example, *conversation types* offer an alternative way to describe sessions, as a set of interactions inside a scope, giving an abstraction of interaction in a common medium. There are also many notions of contract, capturing diverse behavioural properties. Note that contracts differ from simply specifying (say) temporal logical formulae for interactions. They are specified as reciprocal responsibilities among communicating parties, and this global form of contract can be related to individual responsibilities by a process known as *projection*, which is also used in session types for more than two participants.

Thus behavioural types and their associated theories offer a new shape of specification languages centring on interactions and reciprocal responsibilities. But they also enjoy a rich linkage with traditional theories. Contracts are closely related to existing temporal and modal logics, as well as their ramifications. The type structure of a session type is related to an automaton for communicating systems, since it specifies a series of actions among two or more parties. Furthermore, behavioural types give rise to the diversity of type structures cultivated in logics and programming theories, such as polymorphism and dependent types.

All approaches to behavioural types have commonality in that they are closely associated with programming primitives. This is also a key factor for the effective validation of programs against specifications given by behavioural types because the programming primitives offer linkage between programs and type specifications. For example, session types are associated with primitives to initiate a session, communicate through sessions, receive through sessions, and so on. This linkage then leads to a type discipline (a method to validate programs against given types), and its key properties such as principal typing and type/protocol safety (i.e. interactions follow a stipulated protocol), is established. The development of type disciplines closely follows that of traditional typed lambda-calculi and programming languages such as ML and Java. Contracts and logical specifications can then offer richer specifications by elaborating the base type disciplines, which may also make use of spatial localities and their composition for modular specifications.

Many of these studies were initiated in the context of the Sensoria project (Software Engineering for Service-Oriented Overlay Computers), which ran from 2005 to 2010 as part of the EU FP6.

Programming languages and tool development based on behavioural types

These studies give a foundation for the design of new programming languages based on and supporting behavioural types, and for an enhancement of traditional programming languages with behavioural types. From the development viewpoint, the key merits of using behavioural types may be summarised as follows.

- Clean and general structuring principles for implementing, composing and managing communicating distributed systems (distributed services). Given the nature of distributed programming, this aspect covers not only a source program for each endpoint but a configuration of many components, which may be written in different programming languages.
- Suggestions for the design of execution machinery (runtime system) for programming languages for distributed services, through the linkage between programming primitives and behavioural types.

- Simple and intuitive specifications of key properties, which are validated either statically (at compilation time) or dynamically (at runtime), the latter enabling a change of specifications over time. Both static and dynamic validation can make use of efficient algorithms whose correctness is established through the underlying theories.

In recent years, several research projects have started to develop new prototype programming languages based on behavioural types, as well as extensions to existing programming languages. In the latter thread, the languages being extended include Java (the SJ project), ML, BPEL, Scala (the Session Scala project) and C. Microsoft has developed the language Sing#, which uses elements of behavioural types (session types) for systems level programming in the Singularity operating system.

Research prototype languages are often equipped with a full type checker, as well as a lightweight runtime system. Reflecting their experimental nature, runtime and tool support is often elementary. However, through their development, many important results have been obtained regarding, among others: programmability; type checking and type inference algorithms; runtime design; integration of behavioural types with traditional programming language constructs and their validation methods; dynamic validation mechanisms; and design and development experience. Some of these experiments are being further developed to offer more robust production-level support of safety assurance based on behavioural types, often associated with existing frameworks and languages, such as SOA, BPEL, and messaging infrastructure.

B.3 Reasons for the Action

Society is becoming dependent on large-scale distributed software systems, and there is an urgent economic and societal need for improved technologies and methodologies for designing, analysing and building such systems. These technologies and methodologies must be based on a sound theoretical foundation. Many foundational theories and technologies are in place, but delivering them to the software industry and achieving the goal of more reliable large-scale distributed systems will require a co-ordinated effort to ensure a flow of results from foundations to programming languages to applications.

Specific objectives of the Action, at the co-ordination level, include:

- Developing a clear picture of where particular expertise is located, in order to be able to link problems with solutions.
- Supporting Short-Term Scientific Missions, especially for PhD students and early-career researchers, in order to facilitate the development of lasting research collaborations which can be supported by national funding.
- Organising scientific meetings and workshops.
- Maximising the value of existing industrial collaborators by putting them into contact with a wider range of participants in the Action.

The networking activities of the Action will result in a strengthened and consolidated European research community with effective working practices to ensure that foundational research informs and is incorporated into effective programming languages and tools, which are in turn delivered to the software industry for use in practical projects.

B.4 Complementarity with other research programmes

The Action will co-ordinate research that is taking place in around 20 current nationally-funded projects in various European countries.
Three current EU FP7 projects have concerns that are connected with those of this Action to some extent.

- ANIKETOS (Secure Development of Trustworthy Composable Services, 2010 - 2014)

ANIKETOS has a very practical focus on the security of service-oriented systems. It is likely that the Action will be able to offer appropriate theory to support some of the work of ANIKETOS; conversely, the industrial members of the ANIKETOS consortium should be a good source of practical case studies and potential application areas for the results of this Action. It will therefore be important to develop a relationship between this Action and the ANIKETOS project.

- ASCENS (Autonomic Service-Component Ensembles, 2010 - 2014)

The ASCENS project aims to design methodologies to build “ensembles” of components in which there is a strong emphasis on both service-oriented architectures and a high degree of autonomy, self-awareness and adaptability on the part of components. The project includes the development of theoretical foundations and formal verification techniques, and these goals are very much in line with those of this Action. In particular, there is a clear possibility of applying behavioural type theory to the application domains considered by ASCENS. It will be useful to develop mutually beneficial collaboration.

- HATS (Highly Adaptable and Trustworthy Software using Formal Models, 2009 - 2013)

HATS also aims to improve the reliability of large-scale software systems. Its approach is to develop a rigorous foundation for informal software engineering practices based around the concept of software product families. Behavioural specification formalisms, and tools based on them are important aspects of this project, but it contrasts with the approach of this Action: it does not propose behavioural *types* as a new structuring concept for programs. It will be useful to understanding the results of HATS during the initial stages of this Action.

COST Action IC0901 (Rich Model Toolkit - An Infrastructure for Reliable Computer Systems, 2009 - 2013) aims to improve the reliability of computer systems by developing new frameworks for automated system analysis. In broad terms, the goal of improved reliability is similar to that of this Action. The difference is that this Action has a strong focus on developing new programming languages, whereas COST IC0901 develops external tools to be used with existing programming languages. Nevertheless, it will be useful to establish contact with COST IC0901.

C. OBJECTIVES AND BENEFITS

C.1 Aim

The aim of the Action is to use behavioural type theory as a basis for improved programming languages and tools for the implementation of reliable large-scale distributed software systems.

C.2 Objectives

The main objective of this Action is to develop the domain of certified software for global services, providing languages and tools for automatically checking behavioural properties of concurrent and distributed software systems, specified with simple yet expressive type languages. Successful research in this area will have a deep and broad impact on the practice of software development, and on the scientific theories underlying our understanding of distributed computing systems.

The Action has the following sub-objectives:

O1. Co-ordinate European research activity on the theory and application of behavioural type systems, and the deployment of programming languages and tools based on them, organized into the following themes which will each have a Working Group.

- Theoretical Foundations
- Security
- Programming Languages
- Tools and Applications

O2. Build an effective working community of European researchers in this area, by means of the following activities:

- Regular meetings of the Management Committee and the Working Groups
- Annual workshops (following a successful workshop in April 2011)
- Training schools for PhD students and early-career researchers

- Short-Term Scientific Missions between institutions, especially for PhD students and early-career researchers
- Encourage the use of bi-national joint PhD programmes between institutions in the Action.
- Actively seeking to increase the membership of the Action

O3. Encourage the industrial adoption of advanced programming languages and tools, by working with existing industrial contacts and developing new ones.

C.3 How networking within the Action will yield the objectives?

The scientific objectives will be achieved through close collaboration between the participants in the Action. This will include exchange of PhD students and researchers through Short-Term Scientific Missions, and annual workshops that will report on the latest research results and initiate new collaboration activities. The Action will enable PhD students and early-career researchers to develop and share their expertise, and will contribute to creating new connections between established researchers.

The following specific objectives, with quantitative targets where appropriate, are planned. The targets will be used by the Management Committee to monitor the progress of the Action.

At Action level:

1. The Action will involve at least 36 academic and research institutions in 14 countries (30 institutions in 12 countries have already expressed interest in this Action).
2. The activities of the Action will include participation by at least 9 industrial collaborators (6 companies have already expressed interest) and a large-scale academic development project. Several of the industrial collaborators are actively engaged in the deployment of behavioural types technologies in their development projects.

3. At least 6 collaborative, transnational projects on specific topics relating to the Action will be submitted to national and European programmes.
4. Management Committee meetings will be held every 6 months.
5. A project website listing upcoming and past activities, paper abstracts and presentations will be prepared by the Action participants.
6. There will be an annual workshop which serves as a meeting for all Action participants and as a peer-reviewed symposium.
7. There will be a total of at least 100 publications in international peer reviewed journals and conference proceedings as a result of the scientific projects and annual workshops of the Action.
8. There will be at least 120 Short-Term Scientific Missions within the Action.
9. There will be a final report that describes the outcomes and successes of the Action.

At the level of Working Groups:

1. Working Group meetings will be held annually, in conjunction with the annual workshops.
2. Each Working Group will produce a state-of-the-art report during the first year of the Action, and thereafter an annual progress report.
3. Working groups will organise special sessions, for example tutorials or focussed discussion meetings, at international conferences.

C.4 Potential impact of the Action

This Action will co-ordinate European research in the field of behavioural types for concurrent and distributed software systems. It will facilitate collaboration and knowledge exchange between research centres across Europe, thus maximising the benefits of various national research programmes. This in turn will consolidate the internationally leading position of the European research community in this area.

The software industry (including the open source community), for which programming communication and concurrency is one of the largest challenges at present and in the near future, will benefit from improved theories, programming languages, methodologies and tools, which will make it easier to develop correct and safety-assured software systems that rely on concurrency, distribution and communication. This benefit will be seen at a range of scales, from single applications running on multi-core chips through to global service-oriented systems.

Teaching programmes at the participating institutions will benefit from the possibility of offering advanced courses based on the latest research in the topics of the Action. This includes the potential for academic staff to deliver courses at other institutions, based on contacts developed through the Action and taking advantage of existing academic exchange schemes such as Erasmus.

Society will benefit from increased reliability of the essential technological infrastructure represented by concurrent and distributed computing systems.

C.5 Target groups/end users

The benefits of the Action, described in the previous section, will be felt by the following groups and end users.

- The scientific community
- The software industry
- Society as a whole

This Action has been prepared by members of the scientific community, in consultation with representatives of the software industry who are already actively involved in collaborative projects with participants in the Action.

D. SCIENTIFIC PROGRAMME

D.1 Scientific focus

The following specific aims provide a focus for research aimed at addressing the challenges presented by large-scale distributed software systems.

- Develop further theories of behavioural types so that behavioural types wich treat different properties (such as interaction patterns, safety, liveness, information flow) can be uniformly specified, integrated and assured.
- Develop meta-theories of behavioural types, enabling description of the features and relationships of a range of theories without duplication of effort.
- Extend behavioural type theory with quantitative features, to allow specification and analysis of non-functional properties such as response time or resource cost, giving a uniform basis for policy specifications and enforcement including service-level agreements.
- Further develop theoretical and practical understanding of sessions, roles and relationships, generalising from an identity-based view to a role-based view of the behaviour of session participants, so that theories can offer solid foundations of a broad range of programming language methodologies.
- Develop theories, tools and methodologies for the specification and implementation of fault tolerance and error recovery, which are essential in service-oriented distributed systems.
- Further develop theories, programming methodologies and tools based on choreography, orchestration and global types, to specify, analyse and assure properties of the collective interaction among software components.
- Develop theories and tools for the analysis of fairness and liveness properties in service-oriented systems.
- Develop theories and tools for the integrated specifications of security properties including secure information flow in a broad sense, and access control based on behavioural types, so that it enables effective runtime verifications.

- Design, develop and examine the dynamic runtime enforcement mechanism for behavioural types, so that we can specify new properties for distributed services at runtime and they can be enforced, with a formal assurance coming from theories of behavioural types.
- Develop mechanisms for negotiation, adaptation, extension and evolution of behavioural contracts and policies within long-lived environments for service-oriented software.
- Develop new production-level programming languages and programming environments based on behavioural types, building on the preceding experiments on prototype programming languages, with formal support for safety assurance and with emphasis on efficiency and scalability.
- Improve support for service-oriented programming in traditional programming paradigms, in particular object-oriented programming languages.

The Action is structured into four themes, each with a Working Group to co-ordinate its activity. These themes are (1) Foundations, (2) Security, (3) Languages, (4) Tools & Applications. Most of the aims listed above need to be addressed in relation to every theme, and the WGs will communicate to ensure that each topic is treated in a consistent way across all themes. The specific tasks of each WG, described in Section D.2, target these aims. Significant scientific innovations are expected through a close collaboration of the best experts in the field, in unification of foundational theories and their transfer into practical technologies and tools for software development.

D.2 Scientific work plan methods and means

As mentioned above, the scientific work of the Action will be co-ordinated by four Working Groups. Each WG (Working Group) will promote its theme, ensure consistency and avoid duplication of effort, and form links across themes and with industry.

Working Group 1: Foundations

Communication-centred software and service-oriented computing raise new theoretical challenges, involving different research fields and cultural experiences. One of the central issues is the need to specify and validate systems whose components are developed and maintained by third parties who are not necessarily trustworthy, which are dynamically searched and assembled, and which evolve in time. Another central point is the nature of the properties users and administrators wish to ensure for such systems, which are often interactional, reciprocal and global in nature (e.g. developers wish to build a distributed application whose individually developed components interact without deadlock; or users wish to maintain fair and safe usage of a service as a whole).

These problems have been attacked on the basis of several different theories, mainly process algebras, logic and type theory. The main issue of the WG is that of putting on the same footing apparently heterogeneous aspects of software development and verification, including concurrent and distributed programming disciplines, specification and verification of software components, static and dynamic checking of protocol and policy compliance.

The concepts of session, conversation and choreography have emerged as structuring principles for concurrent and distributed systems. Behavioural types merge ideas from process algebras and type theories, so that they provide a common foundation for structured software development and automatic verification, both statically and dynamically, which are well-suited to concurrent and dynamic systems. The tasks of this WG include:

1. To develop the theory of behavioural types clarifying how abstract process descriptions should be used as computational invariants and module interfaces.
2. To face the problem of expressivity of type languages w.r.t. safety, liveness and fairness of systems, in the context of both theoretical calculi and programming languages, investigated by WG3 of this Action, while keeping the effectiveness and efficiency of the verification procedures (including type systems) through which such properties can be ensured.
3. To define a theory of quantitative behavioural types, viewed as an abstract operational model accounting for non-functional aspects of the system's behaviour.

The present state of the art shows a plethora of formalisms, often sharing the same roots but with significant theoretical differences. The WG will aim at developing a common formal framework, allowing for comparison and study of different approaches currently developed by the participating teams. By focusing on the conceptual and technical problems emerging in the activity of the other WGs, WG1 is aimed at the integration and further development of different domain specific approaches, especially with respect to qualitative and quantitative aspects of system behaviour, and the development of new solutions taking advantage of their strengths in the respective domains of application.

Working Group 2: Security
Enforcement of security properties is an important application domain of behavioural types. In particular, analysis of information flow, which is an essential aspect of both data confidentiality and data integrity and requires a dependency analysis, can greatly benefit from the use of behavioural types, since these also allow for a direct description of the causal dependencies in interprocess communication and the order in which data will flow between processes. From a different viewpoint, instead of specifying safety properties for individual, disparate messages, their specifications become much more amenable in terms of specifiability and effective validations when they are given based on protocols.

Applications of behavioural types to concrete case studies and development of type theories that target specific application domains are among the concrete results and unifying themes of the Action. They are therefore also the focus of this WG. The Action will achieve its objectives through the joint development of theoretical foundations and tools.

The activities include:

1. The development of type theories that make it possible to reason about security properties such as secrecy, authenticity, confidentiality and integrity for communication protocols, and the identification of other forms of security properties that are amenable to type-based analyses using new type theories.
2. The development of algorithmic techniques for type-based analyses of such security properties, for both static and dynamic validations.

The type theories and resulting algorithmic techniques should apply both to mathematical models of computation (such as process calculi) and to existing programming and specification languages.

The working group will build on experience gained from the following past successes of its experts:

- leading major efforts in the development of type theories for security issues;

- developing specialized tools for reasoning using types to reason about security issues.

The Action is therefore in a unique position to develop type theories of this kind and to foster collaboration within the community. It is important that WG2 coordinates its activities with the other WGs of the Action; this will happen through joint meetings and via the involvement of working group members in other working groups.

The infrastructure of WG2 will consist of a repository of type theories for security properties. These type theories will be applied and validated by tool development and case studies within WG4.

Working Group 3: Languages

The study and development of behavioural type theories aims at the integration of behavioural types into mainstream programming languages and at their use as a guideline to design new languages that incorporate them. The working group focuses on research in which we develop theories, principles and key runtime mechanisms for programming with behavioural type systems as intrinsic structures. Actual development of programming environments and tool chains which can be used by developers is covered by WG4 described below. The Languages working group will pursue two distinct lines of research, aimed at the integration of behavioural types in:

1. Mainstream, general-purpose languages
2. Domain-specific languages.

Concerning mainstream languages, it is necessary to consider two orthogonal aspects, namely the programming paradigm and the degree of integration of behavioural types within the languages.

Regarding the former, this WG will target different programming paradigms, including, but not restricted to, object-oriented languages and functional languages. As programs are meant to manipulate data, of particular interest to WG3 are the XML (Extensible Markup Language) and JSON (JavaScript Object Notation) data formats.

- Object-oriented languages are relevant for their widespread adoption in the current development of software, for the wealth and popularity of tools that are available, and because objects nicely fit a distribution model to which behavioural types can be applied naturally.
- Functional languages are relevant for their qualities of being easily endowed with high-level type-theoretic and concurrent extensions, for their natural support to parallelism, and since they permit rapid prototyping.
- XML/JSON/YAML (YAML Ain't Markup Language) data formats are interesting for the role of XML and JSON as the de facto standard exchange paradigms in distributed, service-oriented computing, and need to be integrated as part of description languages for protocols and other global properties.

Within each of these programming paradigms, the WG will investigate the following, increasingly invasive ways to integrate the support for behavioural types:

- By means of explicit code annotations in the form of comments or pragma directives. In this way, (fragments of) behavioural type disciplines can be plugged into existing languages without requiring unreasonable changes to their syntax and semantics. Tools can then be developed for analysing and processing explicitly annotated code, in a similar way to what currently happens with many integrated development environments (IDEs) for programming. In order to minimise the amount of information provided by the programmer, it may be desirable to confine the use of explicit annotations to APIs dedicated to communication and synchronisation, and to devise suitable type inference techniques for automatically deriving type information in code that uses the APIs.

- By means of syntax extensions to be handled by pre-processing. In this way, high-level programming constructs (for communication, synchronisation, coordination) are more easily associated with high-level type information than the corresponding implementations in terms of low-level communication primitives.
- By means of conservative extensions of existing languages. Extensions of core research-oriented languages such as, for object-oriented programming, Featherweight Java, will be considered first. Then, results will be validated in fully-featured languages.

Regarding the first point, note that the applicability of behavioural types to traditional languages includes both statically and dynamically typed languages; in the latter case, for example Python, through runtime verification.

Concerning domain-specific languages (DSLs), the WG will follow two distinct research tracks:

- The adoption of behavioural types within existing DSLs designed for application areas and use-cases to which behavioural types naturally apply, such as service-oriented programming, distributed security protocols, distributed queries for semi-structured data.
- The development of new domain specific languages explicitly targeted at exploiting the characteristics of behavioural types and aimed at the coordination and/or orchestration of wide-area distributed services.

Working Group 4: Tools & Applications
This WG will co-ordinate work in two directions.

1. The development of software tools including programming language environments, as a linkage between behavioural type theories and development practice for large-scale distributed systems.
2. The organisation of case studies to apply theories, languages, methodologies and tools developed throughout the Action, to the practice of large-scale software development, centring on the above mentioned software tools.

The aim is to develop tools with which developers can build, analyse and maintain systems incorporating behavioural types. They include analysis tools that are applicable to already running applications built with standard languages and development methods e.g by automatically deriving and annotating code with type information in communication APIs as described in WG3 above. They also include tools for extensions of existing languages and new domain specific languages developed in WG3. Concrete examples of such software tools include:

- A concrete embodiment of developed programming principles as concrete programming environments. This includes, if it is a new programming language, production-level language processors (parsers) and runtime; or, if it is for existing programming languages, as APIs and runtime libraries. These environments include bindings to major communication transports such as TCP (Transmission Control Protocol), UDP (User Datagram Protocol), HTTP (Hypertext Transfer Protocol), AMQP (Advanced Message Queuing Protocol) and ZeroMQ, as well as messaging formats such as XML, JSON and Google protocol buffer.
- A tool chain in which a developer can specify contracts for inter-component communications, project such a contract onto each endpoint through projection algorithms, and, through multiple language bindings, use these endpoint constraints to validate source code written in a concrete programming language, to ensure that, when these components are executed and interact with each other, they conform to the original contract.
- A complementary tool chain for regulating behaviours of distributed applications, which enables specifications of behavioural constraints at various levels, from bare protocols to more complex policies. This will generate data for runtime monitoring of the interaction behaviour of applications, in order to check compliance with the specified contracts.

The central point of having these software tools is to make it possible for developers to effectively benefit from the safety and liveness guarantees for the constructed software systems provided by the theories of behavioral types without having to deal with the complexity of understanding the theories themselves.

Regarding point (2) above, we will first identify the main strengths and application areas of the tools developed in research projects lead by members of the Action. This effort will produce a suite of case-studies emphasising the benefits of the various tools, at the same time contrasting and comparing them. Afterwards, we will organise use-case explorations, development/industrial experiments and feedback from industry, through the contacts that have been cultivated by participants in the Action during their own nationally-funded research projects during the last several years. Involving these development partners in the Action, via the activities of this WG, will maximise the benefit for them and for the Action, by putting them into contact with a broader group of researchers that will provide valuable feedback on each existing application and/or tool. The programming languages developed or adapted by WG3, and the tools and software development environments developed by this WG, will be channelled towards the industrial partners for trial in suitable case studies and large-scale explorations. Conversely, the requirements of application domains such as healthcare, financial services, e-science and development of large-scale distributed software infrastructure, will be fed into the activities of all WGs in order to maintain a focus on the needs of end-users.

E. ORGANISATION

E.1 Coordination and organisation

The Action will be co-ordinated by the Management Committee (MC), which as usual will have a Chair, a Vice-Chair and representatives from the participating countries. The scientific work of the Action will be overseen by four Working Groups (WGs), as described in Sections D.1 and D.2. Each WG will have a co-ordinator who will participate in the MC meetings in order to report on progress. There will also be a website manager and a co-ordinator of Short-Term Scientific Missions (STSMs).

Management Committee

The main responsibilities of the MC are:

- election of the Chair and Vice-Chair, and appointment of the WG co-ordinators, the website manager and the STSM co-ordinator
- ensuring that the planned activities of the Action (MC meetings, WG meetings, STSMs, workshops, training schools) take place in such a way as to meet the objectives of the Action
- producing the annual reports and final report of the Action
- ensuring that the activities of the WGs are co-ordinated
- ensuring that the scientific results of the Action are disseminated appropriately, including via the website

The MC will have an annual meeting, which will be part of the annual workshop of the Action (see below). The website of the Action will have a discussion forum for the MC, which will be used for continuous discussion of the progress, direction and activities of the Action. At half-year points, between the annual meetings, the MC will have a tele-conference for more formal discussion of any necessary business.

Working Group Meetings

Each WG will use a continuous process of discussion, led by its co-ordinator, in order to achieve its objectives. This discussion will take place in a forum on the Action's website, by email, and by Skype audio/video calls. Also, at the annual workshop of the Action, there will be discussion sessions related to the theme of each WG, and a more formal meeting of the members of each WG.

Workshops

There will be an annual workshop of the Action, at which all participants will meet to exchange ideas, report on progress and plan further collaboration. The workshops will also be open to participation by any interested people.

Training Schools

The Action will organise two training schools, in Years 2 and 4. These will be aimed mainly at PhD students and junior researchers. Courses will be contributed largely, but not exclusively, by participants in the Action.

Short-Term Scientific Missions

STSMs are an important part of the activity of the Action. They will consist of short visits, typically of one or two weeks but exceptionally of longer duration, between participating sites. STSMs are primarily intended for PhD students and early-career researchers to visit other research groups in order to acquire new expertise or to contribute their expertise to particular projects. They will facilitate the exchange of ideas between participating institutions and will help PhD students and early-career researchers to develop contacts and collaborators which will be of benefit to their later careers.

Website

The website of the Action will be set up at an early stage, and will be used extensively for discussion in a collection of forum areas for the MC and each WG. The website will also be essential for dissemination of the results of the Action, as described in Section H.

Milestones

The reports produced by the WGs and the MC, and the published material from the annual workshops and the training schools, constitute a series of milestones for the Action. Specifically:

- M1 (end of Year 1): State-of-the-art reports by each WG. First Annual Report by each WG. First Annual Report of the Action. Published proceedings of the first workshop.
- M2 (end of Year 2): Second Annual Report by each WG. Second Annual Report of the Action. Published proceedings of the second workshop. Published tutorial material from the first Training School.

- M3 (end of Year 3): Third Annual Report by each WG. Third Annual Report of the Action. Published proceedings of the third workshop.
- M4 (end of Year 4): Final report by each WG. Final report of the Action. Published proceedings of the fourth workshop. Published tutorial material from the second Training School.

These milestones will enable the MC to monitor the progress of the Action towards the achievement of its objectives, and to plan changes in the work programme if necessary.

E.2 Working Groups

The Action will have four Working Groups, which are described in Sections D.1 and D.2. A co-ordinator for each WG will be appointed by the MC. Membership of the WGs will be determined by the interests of the participants in the Action. It is expected that many participants will belong to more than one WG, and this will help to ensure compatibility between the directions being taken by the WGs.

E.3 Liaison and interaction with other research programmes

The Action will liaise with the EU FP7 projects mentioned in Section B.4. Initial approaches will be made by the Chair of the Management Committee and the co-ordinators of the Working Groups. Representatives of those projects, as well as others that may be identified during the course of the Action, will be invited to participate in relevant meetings and workshops of the Action. It is expected that the greatest benefit of interaction with the projects mentioned in Section B.4 will come from access to a wider range of practical case studies and scenarios.

The Action will also develop links with COST Action IC0901, also mentioned in Section B.4. The open and extensible nature of COST Actions means that mutual participation in workshops and working groups should be straightforward.

E.4 Gender balance and involvement of early-stage researchers

This COST Action will respect an appropriate gender balance in all its activities and the Management Committee will place this as a standard item on all its MC agendas. The Action will also be committed to considerably involve early-stage researchers. This item will also be placed as a standard item on all MC agendas.

It is well known that computer science does not have a good gender balance. This applies to every level from undergraduate courses through PhD students and junior researchers to university faculty members and senior researchers. However, the scientific community proposing this Action has a better gender balance than computer science as a whole. The list of experts contains 12 female members from a total of 64, i.e. just under 20%, which compares favourably with a typical ratio of 15% or less among the faculty members of many university departments. These female members include early-stage researchers, mid-career scientists and senior researchers/professors. There are also a significant number of female PhD students among the research groups participating in the Action. Building on this strong initial position, the Action will ensure that female participants are actively involved in all areas, including membership of the MC and the WGs, participation in Short-Term Scientific Missions, presentation of results at the annual workshop, and participation in the Training Schools as both lecturers and participants.

The list of experts also contains many early-stage researchers, and all of the research groups involved in the Action have PhD students who are likely to become post-doctoral researchers during the Action. This Action will ensure that early-career researchers have the opportunity to participate fully in scientific meetings associated with the Action. In particular, short-term scientific missions will benefit early-career researchers. Training schools organised by the Action will benefit PhD students and early-career researchers as participants. Also, when appropriate, early-career researchers will be invited to present courses at training schools, which will benefit their career development. Similarly, active involvement in the WGs and in preparation of their annual reports will be valuable experience for early-career researchers.

F. TIMETABLE

The duration of the Action will be 4 years.

The Action will begin with a kick-off meeting. The MC will appoint co-ordinators of the WGs, and the website manager and STSM manager.

There will be an annual workshop for all participants in the Action: years 1, 2, 3, 4.

There will be two training schools, combined with the workshops in years 2 and 4.
There will be an annual meeting of the MC, and annual meetings of the WGs, combined with the annual workshops. The format of a WG/MC meeting will be half a day for the WGs to meet for discussion and for a business meeting, and half a day for the MC meeting (including the co-ordinators of the WGs).

Time (Years)

year	**0**	**1**	**2**	**3**	**4**
Kick-off Meeting	*				
MC & WG Meeting		*	*	*	*
Workshop		*	*	*	*
Training School			*		*

G. ECONOMIC DIMENSION

The following COST countries have actively participated in the preparation of the Action or otherwise indicated their interest: DE, DK, FR, IE, IT, NL, PL, PT, RO, RS, SE, UK. On the basis of national estimates, the economic dimension of the activities to be carried out under the Action has been estimated at 48 Million € for the total duration of the Action. This estimate is valid under the assumption that all the countries mentioned above but no other countries will participate in the Action. Any departure from this will change the total cost accordingly.

H. DISSEMINATION PLAN

H.1 Who?

The results of this COST Action will be disseminated to the following target audiences:

- Participants in the COST Action.
- Researchers in related fields of computer science.
- Software developers within the area of large-scale distributed systems.
- Computing researchers and practitioners more generally.
- The general public with an interest in technology.

H.2 What?

For participants in the Action, the following dissemination methods will be used.

- An internal, protected website with description of completed, current and future activities. This includes working documents, publications, reports from workshops and information about grant applications involving members of the Action
- An internet-based communication network for interaction between participants, with special focus on the needs of PhD students and early-career researchers.
- Short-Term Scientific Missions by PhD students and early-career researchers within the Action.
- Two training schools for PhD students and early-career researchers.
- An annual workshop which serves as a meeting for all Action participants and as a peer-reviewed symposium.
- Publications in international peer reviewed journals and conference proceedings as a result of the scientific projects and annual workshops of the Action.
- Annual reports by the Working Groups and by the Management Committee.
- A final report that describes the outcomes and successes of the Action.

For researchers in related fields of computer science, the following dissemination methods will be used.

- Publication of state-of-the-art reports, annual reports, case study reports, workshop proceedings, software documentation, and the final report.
- A public website with general information about the Action as well as a repository of publications by the Action.
- The annual workshops, and the two training schools, organised by the Action.
- Contributions to other national and international conferences and symposia in the form of tutorials and satellite events.
- Articles in peer-reviewed scientific and technical journals and conferences.

For software developers within the area of large-scale distributed systems, the following dissemination methods will be used.

- Invitation to participate in trials of novel programming languages, tools and methodologies developed by the Action.
- Release of open-source software tools for system analysis, based on the research carried out by the Action.
- Encouraging PhD students within the Action to apply for internships with companies who may benefit from software produced by the Action.
- A public website with general information about the Action as well as a repository of publications by the Action.

For computing researchers and practitioners in general, the following dissemination methods will be used.

- Posting of general information on the public website.
- Less technical articles in general computer science publications such as Communications of the ACM, IEEE Spectrum, and magazines of national computing societies.
- Articles in publications by national funding agencies.

For the general public, the following dissemination methods will be used.

- Articles in general science and technology publications.
- Participation in events aimed at public understanding of science.
- Posting of information and non-technical articles on the public website.

H.3 How?

The dissemination methods used within the Action will be used to present the research insights obtained. Research publications are a traditional form of disseminating knowledge both within a specific research community and to a wider research setting. Apart from this form of dissemination,

- The *website* will have a public part that will be used to present published papers and to support stable citation and the correct attribution of work done. This public part will disseminate knowledge to the wider research community. A password-protected part of the website will allow for communication of ongoing research within the Action in the form of online discussion fora as well as draft manuscripts.
- The *annual workshop* will act as a forum in which members of the Action as well as other researchers can present their work.
- The *exchange visits* by PhD students will, when seem as a means of disseminating knowledge, serve to increase knowledge of work in progress.
- The *final report* will disseminate the results of the Action to the community as a whole.