We are concerned with building predictably dependable computer-based systems, including those using modern component-based and service-oriented architectures.

1. The challenge of achieving predictable dependability
2. Dependability metadata and dependability-explicit computing
3. Examples: the WS-Mediator architecture
4. Dependability-explicit system development
5. Research agenda
1. Dependability, its Attributes and Means

Dependability of a computing system is its ability to deliver a service that can be justifiably trusted.

Dependability attributes: availability, reliability, safety, confidentiality, integrity, maintainability

Development of a dependable computing system calls for the combined utilisation of a set of four techniques (means to attain dependability):

- fault prevention (rigorous design)
- fault tolerance
- fault removal (verification and validation)
- fault forecasting (system evaluation)

These techniques come at a cost (performance, resources, efforts, etc.)

1. Component-Based Systems and Service-Oriented Architectures (SOAs)

Modern (and future) systems

- they are built by component integration
- these systems employ external services
- new services are built by composing existing services
- components are loosely coupled (belong to different organisations, sold as services to use, out of sphere of our control)

A service is a logical manifestation of some physical or logical resources and/or some application logic that is exposed to the network.
1. Component-Based Systems and SOAs

We need to find *cost-effective and dependable* ways of
- using services and components
- employing potentially expensive error recovery and fault handling.

System integrators have *limited knowledge* about components and services. The problems:
- they do not know how dependable they are, for example
  - when and how services and components fail
  - how good or bad they are
  - the best way to use them
- the existing mechanisms (e.g. registries or brokers) do not provide sufficient or up to date information to make decisions.

*Our aim is* predictable, not accidental, dependability.

2. Achieving Predictable Dependability

*Informed decision-making* is central to achieving *predictable dependability*

**Information (Dependability metadata)**
- What information is relevant to making decisions about using "dependability means"?
- How should this information be acquired, maintained and published?
- How trustworthy is it?

**Decision-making**
- Requires mechanisms/calculi for reasoning about dependability metadata.
- When are decisions made?
  - Design time (e.g. building in a recovery mechanism)
  - Run time in a design-constrained way (e.g. using alternates)
  - Run time (e.g. dynamic reconfiguration)
2. Dependability-Explicit (DepEx) Computing

DepEx Computing treats dependability metadata as first-class data
- while developing the system
  - explicitly incorporating dependability-related information into system development
  - annotating design and implementation files with dependability-related metadata which are updated when the files are processed (e.g. when development moves from one phase to another)
  - maintaining this information to reflect system and environment state after deployment
- while running the system
  - exploiting dependability metadata to aid run-time decision-making

2. DepEx Computing: Metadata

The general idea:
- consumers present dependability requirements
- dependability metadata are associated with components/services
- informed decisions are made using metadata to satisfy the requirements

Various types of component/service metadata:

- safety integrity level, failure rates, failure modes, pre- and post-conditions,
- MTBF, availability, response time, resources consumed, component specification, fault assumptions, types of encryption, etc.
2. DepEx Computing: Metadata

Service metadata can be published by different sources:

- service/component provider
- third party (broker, registry, archive, trader)
- even the client side.

They can be provided as a result of the development process (e.g. testing, expert judgments) and/or collected and updated in run time.

It is necessary to ensure consistency in the published metadata (is “length” the same everywhere?).

- Partly the subject of semantic web work on ontologies
- We would need ontologies of dependability concepts
- Ideally backed up by real theories

Composability of metadata of some types is not straightforward.

What can be concluded about the dependability attributes of a composite system if we have some idea about the individual attributes of services/components?
2. DepEx Computing: Metadata

To ensure the required level of dependability of the composite system we establish

• the identities of the services and their dependability characteristics
• the need to employ fault tolerance

Within the fault tolerance domain, we can make decisions about:

• the number of redundant components
• their identities
• the fault tolerance scheme to be used (passive vs active replication, recovery blocks, N-version scheme, self-checking pair, retry, safe stop, etc.)

Metadata to be used for making decisions about fault tolerance

• Reliability/availability of individual components
• Reliability/availability of replica components (mirror copies)
• Location of components
• Reliability/availability of diverse versions of a component (there are often a variety of “similar” components in the Internet)
3. DepEx Computing: the WS-Mediator Architecture

WS-Mediator (research.ncl.ac.uk/ws-mediator/):
- A fault tolerance architecture improving dependability of Web Services composition from the end-user’s perspective
- Compatible with the Web Service interoperability specifications
- Can be easily integrated into the existing applications
- Provides flexibility and scalability for different application scenarios
- Implements dependability-explicit dynamic reconfiguration
- Simplifies Web Services application development, (e.g. in e-science and e-business integration)
3. DepEx Computing: the WS-Mediator Architecture

**Dependability metadata:** availability, round-trip response time, type of failure

Location-specific dependability monitoring and assessment

Local metadata DBs at the WS-mediator sites

DepEx dynamic service integration (late binding)

DepEx policy-driven reconfiguration (individual and global policies)

**Fault tolerance** mechanisms employ service and path diversity

- Service Alternatives (based on the Recovery blocks)
- N-version Programming (tailored for Web Service composition)
- Multi-routing
3. DepEx Computing: the WS-Mediator Architecture

WS-Mediator framework deployment:
- PlanetLab Overlay network
- Linux: Fedora 2, Fedora 6
- Sun Application server 9 + Glassfish V2

Dependability monitoring and assessment
- Generic Public Web Services
- GOLD virtual organization Web Services

Application of Java WS-Mediator
- BLAST Web Services deployed by
  - the European Bioinformatics Institute (EBI), UK
  - the DNA Databank (DDBJ), Japan
  - Virginia Bioinformatics Institution (VBI), USA
3. An example: Experiments using the Recovery Block Mode

Details of the experiments
- Recovery Blocks mode for the BLAST Web Services
- Aims to automatically select the most desirable BLAST to invoke
- Aims to automatically choose the time-out according to metadata

Fault assumptions:
- Server/service/network:
  - Omission failure, Timing failure, Response failure, Crash failure

3. Experiments using the Recovery Block Mode
3. Experiments using the Recovery Block Mode

Service Alternatives

3. DepEx Computing: other studies

- Bioinformatics workflows, Virtual organisations
- Dependable WS upgrade:

Modes:
- Parallel execution for maximising reliability or response time
- Parallel execution with dynamically changing reliability / response time
- Sequential execution for minimising server load
4. DepEx and Dependability Means

*DepEx can be effective in improving all four dependability means:*

- fault prevention (rigorous design)
- fault tolerance
- fault removal (V&V)
- fault forecasting (system evaluation)

The main focus of this talk is on fault tolerance, which is implemented through error detection and error recovery.

*DepEx facilitates* error detection and allows an effective and predictable use of redundancy during error recovery.

4. Dependability-Explicit Development Model

The dependability-explicit development model was proposed by M. Kaaeniche, J. P. Blanquart and J.-C. Laprie (LAAS, France).

This model distinguishes between *three main classes* of development processes:

- **the system creation process**, which follows the classical development steps
- **core dependability processes**, such as fault prevention, fault tolerance, etc.
- **additional processes** (quality assurance, certification)

The model describes the key activities to be carried out during various development phases.
4. Dependability-Explicit Development Model

**Definition of fault assumptions is a key to definition of all processes**

The fault tolerance process

- at the **requirement phase**:
  - identifies the set of undesirable events (originating in the environment or in the system)
  - defines acceptable degraded modes
  - describes dependability specification
- at the **design** phase defines the architecture, i.e. several levels of abstractions:
  - fault tolerance structuring, redundancy, protection techniques, error detection, error recovery, reconfiguration

4. Dependability-Explicit Development Model

The focus of this talk is on showing

• how dependability metadata can be used to realise this model (the metadata-centric approach)
• how to apply this model in the context of component system integration and, more specifically, in the SOA context
• how to make dependability in general and fault tolerance in particular predictable
• how to systematically apply this model to make run time decisions about fault tolerance

5. Research Agenda

Making dependability-explicit computing reality is a long-term goal. There are many issues to address.

All development information files (starting from requirements):
  specifications, models, program code, data, schemas, files, views, etc.
should be annotated with dependability-related meta-information.
This information should be further developed/updated/refined/reasoned about/modelled during all development phases. 
This needs tool support.

This information should be brought to the system execution time to allow the system and its components, such as layers, connectors, services, data stores, sensors, drivers, processes, etc. to have meta-data attached to them. 
This needs middleware.
5. Research Agenda

Dependability metadata
- Acquisition, Maintenance, Trustworthiness, Ontologies, Publication, Infrastructure characteristics

Decision-making
- Calculi for dependability metadata composition
- Necessary component of dependability cases
- On-line reasoning support for dynamic decision-making

Middleware support for
- Lookup, Deployment, Reconfiguration, Metadata processing (reasoning)

Advanced workflow/service online composition solutions to allow
- Working with metadata (as first class data) explicitly
- Reconfiguration (constrained) and Fault tolerance
- Specification of desired component services (presented to brokers)

Schema to define
- Classes of possible run-time decisions
- Fault tolerance policies to apply and Reconfiguration policies
- Degradation modes

Dependability brokers
- Searching and enforcing "policies" with regard to selection and configuration
5. Summary

Predictable dependability
- Needs informed decision-making
- At both design time or/and at run time

Dependability-explicit (dependability-aware) computing
- Dependability metadata should be published and reasoned about

Research
- Acquiring and publishing trustworthy metadata
- Applying and implementing calculi for metadata
- Support for decision-making
- Dependability brokers and schemas to define dependability policies

Recent relevant papers

A. Gorbenko, V. Kharchenko, P. Popov, A. Romanovsky. Dependable Composite Web Services with Components Upgraded Online. Architecting Dependable Systems III, pp. 92-121. LNCS 3549. 2005