

Acronym: PyPy



Researching a Highly Flexible and Modular Language Platform

and

Implementing it by Leveraging the Open Source Python Language and Community

Acronym: PyPy

Proposal Part B

Proposal Full Title: Researching a Highly Flexible and Modular Language Platform and Implementing it by Leveraging the Open Source Python Language and Community

Proposal acronym: PyPy

Date of preparation: October 2003, updated April 2004

Type of instrument: Specific Targeted Research Project

List of participants

Participant no.	Participant name	Participant short name
1 (coordinator)	DFKI	DFKI
2	University of Southampton	USH
3	AB Strakt	Strakt
4	Logilab	Logilab
5	ChangeMaker	CM

Coordinator name Alastair Burt
Coordinator organisation name DFKI
Coordinator email burt@dfki.de
Coordinator fax +49 681 302 2235

Contents

Proposal Summary Page	6
B.1 Scientific and Technological Objectives of the Project and State of the Art	8
B.1.1 Problem to be Solved	8
B.1.2 Quantified Specific Objective	8
B.1.3 Current State of The Art	9
B.1.3.1 Interpreter Modularity	9
B.1.3.2 Compilation and Optimisation	10
B.1.3.3 Language Extensions	10
B.1.4 Beyond State of The Art	11
B.1.4.1 Theoretical concepts	11
B.1.4.2 Interpreter Modularity	11
B.1.4.3 Compilation and Optimisation	12
B.1.4.4 Language Extensions	12
B.1.4.5 Distribution	13
B.2 Relevance to the Objectives of the IST Priority	15
B.2.1 Supporting Participation	15
B.2.2 Python as a Language "For Everybody"	15
B.2.3 A Development Platform Suited for Tomorrow's European industry	15
B.2.4 PyPy Builds on the Most Successful Language Designed in Europe	16
B.2.5 Solving 'trust and Confidence' Problems	16
B.2.6 Strengthening Social Cohesion	17
B.2.7 Participation of SME's in High-Level Research	17
B.2.8 Contribution to EC Policies	17
B.3 Potential Impact	18
B.3.1 Contributions to Standards	18
B.3.2 Strategic impact	19
B.3.2.1 Strategic Impact of Open Source	20
B.3.2.2 Potential Impact on the Balance of Trade	20
B.3.2.3 European Dimension	21
B.3.3 Innovation Related Activities	21
B.3.3.1 Management of Intellectual Property	22
B.3.4 Dissemination	23
B.3.5 Exploitation	24
B.4 The Consortium and Project Resources	28
B.4.1 Description of the Partners	28
B.4.1.1 DFKI	28
B.4.1.2 University of Southampton	28
B.4.1.3 AB Strakt	29
B.4.1.4 Logilab	30
B.4.1.5 ChangeMaker	31
B.4.2 Foreseen Amendments in Consortium Structure.....	32
B.4.2.1 Foreseen Future Partners.....	32

Acronym: PyPy

B.4.2.2 Alex Martelli : Best-selling author of Python in a Nutshell . Co-editor of Python Cookbook . ActiveState 2002 Activators' Choice award winner. PSF member, Python language developer, PBF board member. Consultant to AB Strakt, developing the CAPS framework. Also consults for other firms on Python and O-O design, teaching, coding, feasibility studies, interfacing. 1989-2002, Cad.Lab (think3, Inc): innovative component architecture for web-enabling existing GUI- oriented apps; Event Manager, interfacing, proprietary protocols. Taught Computer Programming and Numerical Analysis, Ferrara University. 1981-1989, IBM Research: 3 Outstanding Technical Achievement awards, voice recognition, image processing. Laurea 1980, Electronic Engineering, Bologna University, 100/100 magna cum laude.	33
B.4.3 Sub-Contracting	33
B.4.4 Other Countries	33
B.4.5 Quality of Partnership, Involvement of Users and SMEs	33
B.4.5.1 Roles	33
B.4.5.2 Ability to Deliver	34
B.4.6 Resources to be Mobilised	35
B.4.7 STREP Project Effort Form	37
B.5 Project Management	38
B.5.1 Coordination and Management Team	39
B.5.1.1 Large Scale Projects	40
B.5.1.2 Financial Tracking in Projects	41
B.5.1.3 Leadership Skills	41
B.5.1.4 Coordination in EU Projects	42
B.5.2 Management Structure	42
B.5.3 Project Meetings	43
B.5.3.1 Team Meetings	44
B.5.3.2 Project Review Workshops ("learning loops")	44
B.5.3.3 "Sprint" Meetings are the Key to PyPy's Technical Development	44
B.5.3.4 Technical Decisions	45
B.5.4 Quality Control of Technical Development	45
B.5.4.1 Additional Quality Procedures	45
B.5.5 Communication and Reporting	46
B.5.6 Management of Knowledge and Intellectual Property	47
B.5.7 Consortium Agreement	47
B.6 Detailed Implementation Plan	48
B.6.1 Workplan Introduction	48
B.6.2 Research and Technological Aspects and Options	48
B.6.2.1 Phase 1	48
B.6.2.2 Phase 2	51
B.6.2.3 Phase 3	53
B.6.2.4 Documentation and Dissemination	55
B.6.2.5 Maintenance	56
B.6.3 Risks in the Project and Steps to Minimise Them	57
B.6.3.1 Phase 1	57
B.6.3.2 Phase 2	57
B.6.3.3 Phase 3	58
B.6.4 Project Planning and Time Table (Gantt chart)	59
B.6.5 Workpackage List	60
B.6.6 Deliverables List	60

Acronym: PyPy

B.6.7 Workpackage description	63
B.6.7.1 Coordination and Management	63
B.6.7.2 Infrastructure and Tools	64
B.6.7.3 Synchronisation with Standard Python	65
B.6.7.4 The PyPy Core	66
B.6.7.5 The PyPy Translation	68
B.6.7.6 Core Optimisations	70
B.6.7.7 Translator Optimisations	71
B.6.7.8 Dynamic Optimisations	73
B.6.7.9 Extend Language with Search and Logic	75
B.6.7.10 Extend Language with Aspects and Contracts	76
B.6.7.11 Embed in Specialized Hardware	78
B.6.7.12 Implement Security, Distribution and Persistence	79
B.6.7.13 Integration and Configuration	80
B.6.7.14 Documentation and Dissemination	81
B.7 Other Issues	83
B.7.1 Ethical Considerations	83
B.7.2 Gender Issues	84
B.7.3 Safety Issues	85
B.7.4 Other Policy Related Issues	85

Proposal Summary Page

Proposal full title:

'PyPy: Researching a highly flexible and modular language platform and implementing it by leveraging the Open Source Python Language and Community'

Proposal acronym:

The proposal acronym is PyPy.

Strategic Objectives:

This proposal directly addresses the Strategic Objective IST-2002-2.3.2.3 - 'Open development platforms for software and services' of the Second Call of the Information Society Technologies Work Programme.

Our primary Technical objective is to build an open run-time environment for the Open Source Programming language Python. This will facilitate the development and production of networked and distributed software systems and services, and embedded software.

Our primary Scientific objective is to investigate novel techniques (based on Aspect-Oriented-Programming code generation and abstract interpretation) for the implementation of practical dynamic languages, breaking the compromise between flexibility, simplicity and performance trade-offs and expanding the range (small-to-large) of directly supportable runtime platforms.

Our primary Methodological objective is to showcase a novel software engineering process, Sprint Driven Development. This is an Agile methodology, providing a dynamic and adaptive environment, suitable for co-operative and distributed development.

Proposal abstract:

Main goals

The project will research and implement a flexible, configurable and fast version of Python, a popular Free/Open Source programming language. Much anecdotal, and some empirical, evidence suggests that Python is one of the easiest programming languages to learn and is one of the languages in which one can code up a given algorithm the fastest. The project aims to bring this highly productive language to a wider range of platforms, devices and users by providing simple means to build a highly customized language version.

The approach

The project will achieve its goals principally by leveraging excellent research and the strengths of Open Source developer communities. In particular, it will explore and refine Sprint Driven Development, an agile rapid development method pioneered by Python communities in Europe. Sprints accelerate an already productive open development process by intensifying communication and feedback cycles for programmers and researchers. This process will also involve an existing non-profit organisation of Python firms to bind business communities into research and implementation of a unique runtime environment. Technically, the project will refine and implement innovative approaches to language implementation, based on object spaces and abstract interpretation.

Concrete results

The new Python will maintain the semantics of the current implementation but will be more portable, more easily customised and extended, more efficient, and it will cover more programming paradigms in a way that respects the simplicity of the underlying language. In addition, the project will document and refine existing open source development models in the Python community. Although targeted principally at Python, all research results and the language platform will be applicable to other Very High Level Languages.

B.1 Scientific and Technological Objectives of the Project and State of the Art

B.1.1 Problem to be Solved

Current language implementations are static and hard to modify by the programmers that use them ("language users"). Even the implementations of Open Source dynamic languages have non-flexible designs crafted by a small group of developers. (This has always been a 'given'. The language designer, or designers, make certain trade-offs in their language implementation, and the language users are stuck with what was decided, even if it does not suit their particular needs.)

With a new flexible architecture, it is now possible to give application developers more flexibility and configurability in their preferred language. This is especially the case for the rising number of specialized runtime environments where small memory footprints, concurrency or real-time features are essential.

Very-high-level languages (VHLL) offer a highly productive development tool. However, whole-program static compilation is often not suitable, so that VHLL are usually implemented by introducing a bytecode and interpretation level, resulting in a loss of performance. This is in contrast to lower-level languages such as C, which offer better performance but greatly decrease productivity.

The architectures of current interpreter implementations are a compromise between initial simplicity, Unix-style portability and performance in a single code base. The resulting monolithic (C) code bases are inflexible and expensive to evolve. Early design decisions concerning aspects such as memory management, object model and layout, security, multi-threading model are deeply tangled in the code and cannot be reverted, experimented with or adapted to specific runtime environments.

In the long run, user pressure tends to shape interpreters towards performance at the expense of other aspects (simplicity, flexibility), possibly culminating in the introduction of a native 'Just-In-Time' (JIT) compiler, adding a significant amount of code and complexity and further impairing the flexibility. [\[S03\]](#)

Consequently, application developers are often left with bad choices - not only regarding productivity versus performance, but also involving the hard-wired characteristics built into the languages. They would prefer to adapt and configure a VHLL to suit their needs while at the same time developing on top of a customized but standard language offering both productivity and performance.

B.1.2 Quantified Specific Objective

The aim of the PyPy project is to research and implement an interpreter and runtime environment for the Python language, built on a unique architecture enabling both productivity and performance. The main building block is the concept of an Object Space which cleanly encapsulates computational operations on objects, separated from the bytecode interpreter. A number of features, most of which are hard to implement as application-level libraries, can become part of the language

Acronym: PyPy

in a manageable and language-user-configurable way if they are implemented modularly as Object Spaces. For example:

- choice of memory and threading model
- choice of speed vs. memory trade-offs
- pluggable adaptive implementations of various VHLL types, e.g. dictionaries
- distributed/parallel execution (SMP or Networked)
- orthogonal persistence
- pervasive security support
- logic and aspect-oriented programming

The second key idea to reach this level of flexibility is to write the interpreter and Object Spaces in a VHLL language (Python itself) and recover performance with a separate translation process producing specialized efficient low-level code. The translation is not one-shot: it can be repeated and tailored to weave different aspects into releases with different trade-offs and features, giving the user of the language a real choice.

The PyPy project plans to deliver a compliant language implementation passing all unit-tests of the current reference C-implementation that are relevant to the new one. At least 90% of existing unit tests will be directly applicable*. *—Moreover, we will be able to add techniques such as JIT compilation without making the interpreter more complex. Not only will we thus produce a more easily-understood optimising compiler, of particular interest to educators, but we will also get speed increases of 2-10 times over the reference C-implementation. We expect algorithmic code to run at least at 50% of the speed of pure, optimized C code.

B.1.3 Current State of The Art

B.1.3.1 Interpreter Modularity

Haskell monadic modular interpreters [[LHJ95](#)] [[H98](#)] are a researched attempt at achieving modularity for interpreters. They have not been tried on something as large as Python but in the context of Domain-Specific Languages (DSLs). However, the approach is hard for programmers accustomed to more conventional Object-Oriented (OO) Programming to grasp, and requires sophisticated partial evaluation to remove the monadic overhead. On the positive side, monad transformers are powerful enough to modularize continuation passing, exceptions and other control flow aspects.

Some of the kind of modularity we are interested in for interpreters - subsetting, choice among implementation of basics aspects (memory management,...), feature selection - has some similarity with recent research on OS modularity and software composition, e.g. the Flux Group OSKit project and their composition tool Knit [[RFSLE00](#)].

In its basics, the approach of writing a language interpreter in the language itself (a subset thereof) and then producing a running low-level code version through a translation process has already been taken e.g. for the Scheme (Scheme 48) [[K97](#)] and the Smalltalk (Squeak) [[KM97](#)] languages. These approaches are typically based on translation from high-level source code (or parsed source code)

* The reference C-implementation contains some tests that depend on implementation details. The exact line between a language feature and an implementation detail might at times be hard to draw precisely, but in any case only concerns a minority of the tests (less than 10%).

into C. It would be possible within this current state of the art to obtain an interpreter comparable with the current C Python implementation in functionality; but note that our approach already differs in three respects (as explained in more details in the section *Beyond the State of the Art*): what we will translate is a dynamically loaded and initialized program state instead of the static source code; the translator works by abstract interpretation (also known as symbolic interpretation), which makes it independent from the actual source or bytecode; and we will put the key modularity and flexibility concerns into the translator.

We plan to exploit the gained flexibility of the translator much further. It will enable separation on concerns between the translator, the core interpreter, and the Object Spaces, in the spirit of Aspect Oriented Programming (AOP) as developed in [\[KLM97\]](#). AOP separate cross-cutting aspects into separate components. This approach has however never been used on interpreters for large practical languages.

B.1.3.2 Compilation and Optimisation

JIT compilers have been reasonably well studied; an account of their history is given in [\[A03\]](#). But actually writing a JIT compiler for a given language is generally a major task [\[WAU99\]](#). For example the recent Jalapeno Java VM [\[AFGHS00\]](#) and its compilers, although being in written in Java, are a complex architecture, requiring fine tuning, using runtime sampling to assess at runtime the dynamic call graph to drive inlining.

Different techniques to ease this path have been recently explored:

- To implement a dynamic language in a flexible way, it can be written on top of another one, e.g. as a dynamic translator that produces bytecodes for an existing virtual machine. If the virtual machine is already equipped with state-of-the-art JIT compilation, it is possible to leverage its benefits to the new language. This path is not only much shorter than designing a complete custom JIT compiler, but it is also easier to maintain and evolve. This idea is explored in [\[WAU99\]](#) with experimental Java and Smalltalk implementations, built on top of the Self virtual machine. As pointed out in that paper, some source language features may not match any of the target virtual machine features. When this issue arises, we are left with the hard problem of refactoring an efficient JIT compiler-based virtual machine.
- Using a completely different approach, to make it easier to derive a JIT compiler from an existing C interpreter, DynamoRIO instruments the execution of compiled programs and optimizes them dynamically. It has been extended with specific support for interpreters. With minimal amounts of changes in the source of an interpreter, it can significantly reduce the processor-time interpretative overhead [\[S03\]](#). While this offers a highly competitive gain/effort ratio, performance does not reach the levels of a hand-crafted JIT compiler.

The current Python C implementation (CPython) is a straight-forward bytecode interpreter. Psyco [\[R03\]](#) is an extension to it implementing a highly experimental [\[APJ03\]](#) specializing JIT compiler based on abstract interpretation. In its current incarnation it is also a delicate hand-crafted piece of code, which is hard to maintain and not flexible. But this should not inherently be the case. Moreover it would likely benefit from integration with type-feedback techniques such as those developed for Self [\[HCU91\]](#) [\[HU94\]](#).

B.1.3.3 Language Extensions

The family of languages based around logic and constraint programming has proved very useful for knowledge representation and optimisation problems. The key features of these languages are the use of a *logic variable*, which allows the matching of data through unification or constraint processing, and non-deterministic control structures. Work initiated at the DFKI has examined ways to combine these constructs with those from other languages [Sch02]. A key concept here is *encapsulated search*, which hives off these features from the more mainstream programming constructs.

Several people have looked at Python as a programming language for Artificial Intelligence. Logilab initiated a project to implement constraint programming in Python [Log], and one of most popular text books in Artificial Intelligence has Python code for the examples [RN02]. Moreover, the main inventor of the world wide web developed the first inference engine for the semantic web in Python [Cwm]. However, none of these projects has realised code that can be used in production environments. It is simply too slow.

B.1.4 Beyond State of The Art

B.1.4.1 Theoretical concepts

- We present a novel interpreter architecture based on the separation between Bytecode interpretation and **Object Spaces**. The former, i.e. the interpreter's main dispatch loop, handles control flow and supporting data structures (frame stack, bytecode objects...), while each individual operation on objects is dispatched to the Object Space. In effect, an Object Space explicitly captures the notion of "object library".
- Object Spaces simultaneously capture the notion of "abstract domains": indeed, building on this architecture, **abstract interpretation** based on the *same* Bytecode interpreter with a different Object Space gives us unprecedented power and simplicity for all kind of source analysis tools. In particular, this is a new way to efficiently close the gap between a Very High-Level Language (VHLL) and low-level code, including for advanced JIT compilers.
- Many aspects of the interpreter are not fixed in all their details within the interpreter source: they are **translation aspects**, i.e. they are weaved into the low-level translation of our VHLL source by the process of translating it into a low-level equivalent. This allows low-level aspects like multithreading, memory management, and continuation management to be kept orthogonal from the rest of the source.

B.1.4.2 Interpreter Modularity

Object Spaces, in contrast to monadic interpreters, will allow customization with OO techniques. They encapsulate the object operation semantics, creation and global state of all objects. As mentioned above, monad transformers have been used to modularize continuation passing, exceptions and other control flow aspects; the theoretical solution we provide offers a more practical and scalable approach to the modularization of these aspects.

In contrast to the related work previously cited, we don't expect to encode a fixed single interpreter in all its details in a subset of our VHLL (Python). We plan to exploit the flexibility and abstraction gained by using the VHLL and -- most importantly -- the indirectness of translation in order to

Acronym: PyPy

"weave" aspects such as continuation passing, memory management, object layout, threading model etc. at translation time.

Many of these aspects are really cross-cutting concerns in the original Aspect Oriented Programming (AOP) sense. E.g. we expect to code addition between Python integers in a high-level way independent of memory management and of boxing issues. In general our approach relates to the seminal AOP ideas of [KLM97]: the subset of Python in which we express the core interpreter and Object Spaces is the *component language* in the terminology of the paper, while the translator is a *weaver* written with the full dynamism of Python.

In summary, we will explore for each feature and extension how to best implement it by separation of concerns: either in OO-customized Object Spaces, or in the core or in modules to be translated, or as a pluggable behaviour of the translation itself.

B.1.4.3 Compilation and Optimisation

The front-end of the translator itself will be innovative in that it is based on abstract (symbolic) interpretation. The translation of code from our Python subset (in particular the source of PyPy itself) will thus be driven by PyPy's own Python interpreter, made symbolic by plugging a custom Object Space. This turns the Object Space operations into the semantics units of translation as well, leveraging the coherence of our architecture and gaining independence from surface issues including the details of the syntax or of the bytecode itself. Moreover, we can use the full dynamism of Python at system definition time, i.e. when PyPy loads, until we reach a "stable-and-ready" state that the translator can freeze into a snapshot containing exactly the features and modules that we need. In previous work, the translator generally operated on the static source code.

During the translation process, weaving custom aspects will be done mainly as intermediate stages, while the customizable back-end generates low-level code in various styles as needed. For example, it can be in continuation passing style (CPS), or it can target altogether different runtime environments like Java -- providing a cheap way to regenerate a replacement for Jython, the Python interpreter written in Java, without the burden of having to maintain it synchronized with CPython.

Another key innovative aspect of our project is to generate the JIT compiler instead of writing it manually. Indeed, although Psyco was hand-written, large parts have a one-to-one correspondence to whole sections of the CPython interpreter. This is uncommon for JIT compilers, but Psyco's good results give ample proof-of-concept. (This property can be explicitly related to the DynamoRIO project cited above, which instruments the interpreter itself.) The one-to-one correspondence between parts of CPython and Psyco is as follows: to each expression in the source of CPython corresponds a more complex expression in Psyco, which does instrumentation and optimisations. Our plan is thus to generate automatically as much of the JIT as possible, by changing the translator to generate instrumenting/optimizing C instructions instead of (or in addition to) the normal C instructions that would be the direct translation.

B.1.4.4 Language Extensions

Within PyPy, we will be the first to provide usable constraint programming techniques within a popular VHLL. We will take advantage of the light-weight multithreading that we will introduce to Python to handle the constraint processing efficiently and we will examine the use of object spaces to develop implementations of Python that are specialised for search tasks. Doing this in a modular way through the use of object spaces, represents a novel feature of the project.

Acronym: PyPy

The DFKI has a variety of expertise with the semantic web and plays an active role in several W3C working groups. The work of PyPy will dovetail with the work being carried out within the DIRECT-INFO EU project, where the DFKI is supporting a media analysis application by integrating semantic web constructs into the Python-based Zope application server. It is expected that efficient search integrated into the application server framework will lead to many novel applications.

B.1.4.5 Distribution

We expect to develop automatic interpreter build tools that allow choices about aspects, modifications and extensions to be made by the language user, and that enable advanced programmers to develop and integrate their own aspects and extensions to the language.

We believe that ours is a practical and scalable approach to interpreter modularity, applicable to any language, from general to domain-specific ones. The PyPy implementation should quickly reach the large user base of the current, industrial-strength Python, and could eventually form the foundation of the "next generation" Python implementation commonly referred to as Python3000. The efforts will be focused on actively reaching out non-Python communities, based on:

- the build tools to generate customized versions of Python, attractive to a larger industrial base which is currently outside the scope of VHLLs for reasons like performance, configurability, or end-user device resources;
 - the framework of PyPy's source code, reusable to implement other general or domain-specific languages;
 - the theoretical approach and solutions that we publish.
-

Acronym: PyPy

References

- [A03] John Aycock, "A Brief History of Just-In-Time", ACM Computing Surveys 35, 2 (June 2003), pp. 97-113.
- [AFGHS00] Matthew Arnold, Stephen J. Fink, David Grove, Michael Hind, and Peter F. Sweeney, "Adaptive Optimization in the Jalapeno JVM", In Conference on Object-Oriented Programming and Systems, pp. 47-65, 2000. <http://citeseer.nj.nec.com/arnold00adaptive.html>
- [APJ03] John Aycock and David Pereira and Georges Jodoin, "UCPy: Reverse-Engineering Python", PyCon DC 2003, 9pp. <http://pages.cpsc.ucalgary.ca/~aycock/papers/ucpy.pdf>
- [Cwm] <http://www.w3.org/2000/10/swap/doc/cwm>
- [H98] Paul Hudak. "Modular Domain Specific Languages and Tools". ICSR 98. 1998. <http://haskell.org/frp/dsl.ps>
- [HCU91] Urs Hölzle, Craig Chambers, and David Ungar, "Optimizing Dynamically-Typed Object-Oriented Languages with Polymorphic Inline Caches", ECOOP'91 Conference Proceedings, Geneva, 1991. Published as Springer Verlag Lecture Notes in Computer Science 512, Springer Verlag, Berlin, 1991. <http://self.sunlabs.com/papers/eoop91.ps.Z>
- [HU94] Urs Hölzle, David Ungar, "Reconciling Responsiveness with Performance in Pure Object-Oriented Languages", PLDI '94 and OOPSLA '94 <http://www.cs.ucsb.edu/oocsb/papers/toplas96.pdf/reconciling-responsiveness-with-performance.pdf>
- [IKM97] Dan Ingalls, Ted Kaehler, John Maloney, Scott Wallace, and Alan Kay. "Back to the future: The story of Squeak, A practical Smalltalk written in itself." In Proceedings OOPSLA'97, pages 318--326, November 1997. <ftp://st.cs.uiuc.edu/Smalltalk/Squeak/docs/OOPSLA.Squeak.html>
- [K97] Richard Kelsey, "Pre-Scheme: A Scheme Dialect for Systems Programming". 1997. <http://citeseer.nj.nec.com/kelsey97prescheme.html>
- [KLM97] ([1](#), [2](#)) Gregor Kiczales and John Lamping and Anurag Menhdhekar and Chris Maeda and Cristina Lopes and Jean-Marc Loingtier and John Irwin, "Aspect-Oriented Programming", ECOOP'97 <http://www2.parc.com/csl/groups/sda/publications/papers/Kiczales-ECOOP97/for-web.pdf>
- [LHJ95] Sheng Liang, Paul Hudak and Mark Jones. "Monad Transformers and Modular Interpreters". 22nd ACM Symposium on Principles of Programming Languages (POPL'95). January 1995. <http://java.sun.com/people/sl/papers/popl95.ps.gz>
- [Log] <http://www.logilab.org/projects/constraint>
- [RN02] Russell, Stuart, and Norvig, Peter, "AI: A Modern Approach", Prentice-Hall, 2002, <http://aima.cs.berkeley.edu/python/readme.html>
- [R03] Armin Rigo, <http://psyco.sourceforge.net>
- [RFSLE00] Alastair Reid, Matthew Flatt, Leigh Stoller, Jay Lepreau, and Eric Eide, "Knit: Component Composition for Systems Software", in Proceedings of the Fourth Symposium on Operating Systems Design and Implementation OSDI'2000, 2000. <http://www.cs.utah.edu/flux/papers/knit-osdi00.pdf>
- [S03] ([1](#), [2](#)) Gregory Sullivan et. al., "Dynamic Native Optimization of Native Interpreters". IVME 03. 2003. <http://www.ai.mit.edu/~gregs/dynamorio.html>
- [Sch02] Christian Schulte, "Programming Constraint Services". Volume 2302 of Lecture Notes in Artificial Intelligence, Springer-Verlag, 2002
- [WAU99] ([1](#), [2](#)) Mario Wolczko, Ole Agesen, David Ungar, "Towards a Universal Implementation Substrate for Object-Oriented Languages", OOPSLA 99 workshop on Simplicity, Performance and Portability in Virtual Machine Design, OOPSLA '99, Denver, CO, Nov 2, 1999. <http://research.sun.com/research/kanban/oopsla-vm-wkshp.pdf>

B.2 Relevance to the Objectives of the IST Priority

B.2.1 Supporting Participation

The PyPy project will connect expert researchers, programmers and users at all levels at all times. On the social level we will explore and refine new cooperative ways of researching and implementing advanced highly involved technologies. Building on the large communication network of open source programmers we will explore and refine agile methodologies. We believe that our new **fluid networked way of rapid software development presents a new paradigm how people can work together** .

This is an efficient countermeasure to the perceived European "weak culture of transferring and exploiting university research results" (SWOT table from 'IST Advisory Group: Software technologies, embedded systems and distributed systems: A European strategy towards an Ambient Intelligent environment').

B.2.2 Python as a Language "For Everybody"

The European industry, as well as many developers, are not interested in yet another new language. Instead our project will produce an extremely flexible and performing development platform around the Python programming language. Much anecdotal, and some empirical, evidence suggests that Python is one of the easiest programming languages to learn and is one of the languages in which one can code up a given algorithm the fastest. For these reasons, it is **particularly popular among those people whose main competence lies beyond computer science and for whom programming is a secondary activity** , that is to say almost everyone in the information industry. Using Python itself to develop an innovative Python implementation will allow many people to understand the platform and reuse it in their context. Understanding, teaching and adapting our Python implementation will be much easier than almost any other language implementation.

B.2.3 A Development Platform Suited for Tomorrow's European industry

The EU is interested in **software technologies that are reliable, pervasive, interoperable and can be adapted to accommodate new applications and services** . This is exactly the focus of our project. With some unique approaches to building a highly performing and productive development platform we will further innovation and interoperability.

We will build an extensible Python implementation that can be adapted and configured for almost all runtime environments. It will go far beyond the state of the Art in computer languages, and produce a runtime system which is much better suited for the development and deployment of networked, embedded, and mobile devices than any existing language available today. In doing so it will be compliant with the Python language specification requiring no re-training for the tens of thousands of European Python programmers. To dramatically expand the scope of an already very productive development environment can only have a positive effect on European competitiveness.

B.2.4 PyPy Builds on the Most Successful Language Designed in Europe

Python is the most widely used European-designed computer language. Its development started in 1990, at CWI, Centrum voor Wiskunde en Informatica, the National Research Institute for Mathematics and Computer Science in the Netherlands. This project will strengthen European leadership in the area of innovative language design, and increase world-wide awareness of this fact. We can even do our bit to reverse the brain-drain, as talented European language designers will no longer have to move to North America to be involved in language projects which have more than academic interest.

We believe that PyPy presents a **unique opportunity to bring considerable market- and mind share with respect to development tools and environments back to Europe** . The many individuals and companies involved with the project are deeply entangled with the opensource community and businesses. Python and PyPy in particular provide a viable alternative to American closed source language monopolies, while increasing innovation and competitiveness in European businesses and industry and thus contributing to the greater well-being of all European citizens.

Working with the strong Open Source community to overcome the European weakness with development platforms obviously builds on the strategies as outlined in the SWOT-analysis by the IST work group.

B.2.5 Solving 'trust and Confidence' Problems

The IST thematic priority will contribute directly to realizing European policies for the knowledge society as agreed at the Lisbon Council of 2000, the Stockholm Council of 2001, the Seville Council of 2002, and as reflected in the e-Europe Action Plan.

A main target of IST in FP6 is:

solving 'trust and confidence' problems so as to improve dependability of technologies, infrastructures and applications.

PyPy will contribute to this goal because it is build on the strength of the Open Source community. Open Source programs are more widely trusted than proprietary alternatives because they are more transparent, accessible and customizable. Especially in large Open Source projects all parts of the program are under permanent scrutiny of many interested developers. Because we are deeply connected with the large Python community we are confident that PyPy can take full advantage of this open culture.

Moreover, Python is an extremely readable language. Readability was and remains one of its main design goals. This makes maintaining Python programs substantially easier than similar program in less readable languages. It is the maintainability of computer programs which most directly effects their actual and perceived reliability and encourages intelligent usage.

In our view development platforms should not be proprietary but should **empower the user** to get involved and provide appropriate extensions suiting commercial and research needs. Thus rather than 'trusting the manufacturer because you have no choice' you can 'trust yourself'. This second form of trust is far more durable and useful.

B.2.6 Strengthening Social Cohesion

Social cohesion is strengthened when technological advances are no longer the exclusive domain of a technological or commercial elite, but readily accessible by all members of society. The best way to achieve such a goal is to have the participation of all members of society in the design and implementation of new technological advances. In the field of software, this means more than simply providing programs which are easy to use -- it also means providing languages which are easier for people for whom programming is a secondary activity.

- enabling sustainable growth and improving competitiveness both of large and small businesses as well as the efficiency and transparency of governments.

Governments have been embracing Open Source for some time now. To the extent that they will demand programs developed in an Open Source Language, for reasons of transparency, reliability, and national security they will benefit from the existence of PyPy as an Open Source language choice.

B.2.7 Participation of SME's in High-Level Research

The consortium will have close links with the Python Business Forum, an international trade association of SMEs who develop using the Python programming language, and other SME partners. Since SMEs are the main engines for innovation, growth, and competitiveness in the IT sector, supporting these SMEs, and improving the language they use to develop can have a direct positive effect on European competitiveness. Moreover, synergies can be developed between the SMEs and academia, and SMEs and large industrial players. Disseminating knowledge to SMEs is a primary goal of this proposal, and a major focus of our efforts. They will be poised to fully exploit the new language implementation because they are among its developers, and the beneficiaries of a focused effort in knowledge dissemination.

B.2.8 Contribution to EC Policies

There will be transfer of knowledge from research to industry through the participation of software developing SMEs. Thus the SMEs in the project will benefit from the cutting-edge, high level research results. Since the SMEs are in a hurry to commercialise products which use this research, the academics will see that their research is not wasted -- locked into tiny languages which have little effect outside of the academic community. A specialising Just-in-Time Compiler for Python, designed for the use of networked and embedded systems will have immediate effect in reinforcing European dominance in this demanding competitive field, and thus contribute to the employment of people in desirable jobs in a rapidly growing commercial sector.

B.3 Potential Impact

The successful execution of the PyPy project will deliver a highly productive and flexible implementation of an Object-Oriented, Open Source Very-High-Level programming language (VHLL).

This will impact software development in several important ways.

- The cost of software development will diminish.
- The time to market will be reduced.
- The cost of software maintenance will be reduced.
- The barriers to marketing a product will be lowered.

The development methods of the PyPy project will prove that using Sprints, pair programming and test-driven development results in :

- Broader understanding of the code base among developers
- Rapid developments from ideas to working code
- Sustainable project progress through unit testing

The cost of software development is essentially labour costs, time to market, and costs of software tools. Having a (free) flexible VHLL means problems are solved closer to the abstraction level at which they are formulated, thereby improving the productivity of the individual programmer. Higher productivity, obviously, will decrease labour costs and time to market.

However, the greater impact we expect to have is to produce a version of the language which, being superior to anything which has gone before, will be extremely widely used. To that end, our first target group is the existing community of Python/Jython programmers. See the section on Exploitation where this is discussed at length.

B.3.1 Contributions to Standards

There are currently two implementations of Python in common use. The first one, which we will call CPython, (what the world knows as Python), is a C implementation of the Python Programming language which compiles to its own virtual machine. The second one is Jython, a pure-Python implementation which compiles to the Java virtual machine. There is no ANSI standard (or similar standard) for Python. Right now the de-facto standard for the programming language is 'whatever CPython does'. This is far from ideal, and questions arise, especially from the developers of Jython as to which CPython language behaviours are essential to the Python language itself, and which are mere accidents of this particular implementation.

For example, the garbage-collection behaviour of CPython is implemented by reference-counting, which ensures that an object is finalized as soon as the last reference to it goes away. That would be extremely inconvenient (close to impossible) to implement on standard Java Virtual Machines, which have a deliberately under-specified garbage collector (it can collect anything it pleases whenever it pleases...). In this case, the Jython designers had to obtain an explicit ruling from Guido van Rossum, Python's designer -- who ruled that the behaviour of CPython was 'accidental' in this case, and not intrinsic to the Python language specification.

Acronym: PyPy

Guido van Rossum has expressed interest in giving PyPy the status of 'implementation standard' (executable specification) of the Python programming language. PyPy's Object Space flexibility will be crucial in distinguishing "accidental" from "designed-in" characteristics.

Here is the relevant mail from Guido van Rossum:

Having participated in one PyPy Sprint, I am very happy with this project, and hope to see it going forward. The PyPy team includes some of the best minds in the Python community.

Unlike Perl or Tcl, Python is not a "one-implementation" language, and consequently the formal language specification should have priority over the behaviour of a particular implementation. The PyPy project will reinforce this idea, and can be useful in sorting out ambiguities in the specification. It is even possible that PyPy will eventually serve as an "executable specification" for the language, and the behaviour of one of PyPy's object spaces will be deemed the correct one.

--Guido van Rossum (home page: <http://www.python.org/~guido/>)

In order to do this we will have to submit a PEP. A PEP - Python Enhancement Proposal - is a design document providing information to the Python community, or describing a new feature for Python. The PEP process is designed for Community involvement and participation.

There are two kinds of PEPs. A Standards Track PEP describes a new feature or implementation for Python. An Informational PEP describes a Python design issue, or provides general guidelines or information to the Python community, but does not propose a new feature. If we proposed to make PyPy the reference standard of the Python language, we would, obviously, have to submit a Standards Track PEP.

The complete details of how to write a PEP are themselves an Informational PEP -- PEP #1 [\[PEP1\]](#) in fact.

After circulating through the community, PEPs are reviewed by Guido van Rossum, the language author, and his chosen consultants, who may accept or reject a PEP or send it back to the author(s) for revision. Thus the expression of interest from Guido van Rossum is of extreme significance. It is very likely, if all the goals described in the project are completed, that PyPy will become the standard reference implementation of the Python language. In other words, our problems are technical, and not political. The political will is already there to make PyPy the reference language. We merely need to create it.

B.3.2 Strategic impact

PyPy will have a significant strategic impact throughout the IT sector. It produces immensely useful, practical results which shall be immediately exploited by Python developers world-wide. While it addresses issues which have hitherto mostly remained the special province of academia, and significantly enhances the State-of-the-Art, it is not a project that will only satisfy intellectual curiosity.

We intend to make a new reference version of the Python Programming Language, which is faster, more flexible, more extensible, and which gives more control to the individual programmer as to how it is deployed. For instance, you will be able to build a Python interpreter customized for a very small amount of available memory. Or one with speed enhancements only possible because there is

Acronym: PyPy

a huge amount of memory available. Or a PyPy interpreter which will be executed on several machines but offer a single distributed computation space and balances the load by moving execution threads around. We can produce Object Spaces which implement Logic Programming, Aspect Oriented Programming and Design By Contract, hot new topics in computer science research which are rarely seen in industry because existing popular languages do not support them.

An implementation of the language with substantial improvements will have an immediate direct effect on European competitiveness. Moreover, the planned improvements directly target the hand held, mobile, and embedded device sectors, where Europe is the acknowledged world leader. People working in such industries have long desired a high level language with a very small footprint. The new innovative concept of Object Spaces, pioneered by PyPy makes possible the construction of tiny Object Spaces, suitable for running on the smallest devices. Indeed, it will be possible for application programmers to configure Python runtime environments to suit their particular hardware characteristics -- for instance, a version that runs in a minimal amount of memory, or a version that can exploit a huge amount of memory to achieve the highest performance speed.

Python with greater speed will seamlessly improve the offerings of those European companies who already develop using Python. Moreover, many companies resist using Python because of speed concerns. If execution speed, rather than development speed is of paramount importance, then Python is currently not a very good language choice. A faster Python would be appealing to such companies, perhaps appealing enough to motivate them to switch development languages. Already many companies are using Python for their scripting needs, indicating that the speed factor is very significant.

B.3.2.1 Strategic Impact of Open Source

Open Source has now reached an installed application base sufficient to become widely recognized as a viable business standard, especially in Europe. This is in part because the proprietary alternatives are intellectual properties of large USA based companies, but also due to a growing awareness of the other benefits that Open Source can provide. The European Union, comprising heterogeneous distinct regions with different availability of economic resources, is well positioned to take advantage of the Open Source momentum. One reason that Open Source is becoming so appealing is due to its equal suitability for projects based on diverse capital budgets.

Python is an easy to learn, easy to use, Open Source programming language. It is readily accessible to a broad user base, estimated at 175,000 programmers world-wide, employed in education, government and commercial enterprises of all sizes. This project aims at building upon the inherent strengths of Python to ensure its longevity in the commercial and research marketplaces. This will maximize the return of the existing and future capital investment in this technology and ensure Python's widespread acceptance as one of the most cost-effective technology platforms available.

B.3.2.2 Potential Impact on the Balance of Trade

One of the greatest threats to European competitiveness is its dependence upon proprietary closed source software, mostly made in the United States. This is not only an issue of money being spent in the United States is money that is not being spent here, although that affect matters as well. There are two more serious risks.

The first is a threat in the present. Any company which writes its software in a proprietary, closed source language is dependent upon its software provider. If you have a bug, you must wait for them

Acronym: PyPy

to fix it. If this bug is not a high priority for them, you can wait a long time. If you have access to the source you always have the option of fixing it yourself, or hiring somebody else to do that. But this is not the greatest of your worries. The second threat is that you are at constant risk of having your software provider discontinue support for your platform. This is a real threat, not a theoretical one. In 2002, Microsoft announced that it would no longer be supporting Visual Basic 6.0 after the year 2005. All Visual Basic Developers have been told to convert their code to run under Microsoft's new .NET framework. Before that, in 2001, Microsoft suddenly stopped supporting its Visual J++ language platform, meant to be a direct competitor for Java, after settling a lawsuit with Sun Microsystems. No migration path was specified. Microsoft is making these decisions because they make business sense for Microsoft, regardless of the effects on European software developers.

Right now Python is the sixth most popular programming language in the world. Java and Visual Basic (VB), ranked 1 and 2, are closed source proprietary American products. The number one reason that is cited by Java users as to 'why they don't use Python' is that it is too slow. PyPy will fix this. The Visual Basic programmers are in a more interesting position. Microsoft, in its wisdom, has decided to end support of their current platform, Visual Basic 6.0. After the year 2005, these programmers will have to move to Microsoft's .NET. They're understandably unhappy, and tempted to move, not to .NET, but to an Open Source language, just so that they can have control over their own destiny and indicate their displeasure with Microsoft at the same time.

This is a tremendous opportunity for us. Every Java and VB user that switches to Python does not pay license fees to Sun or Microsoft and helps the European balance of Trade.

Of course, all the PBF members are predicting that an improved Python will improve their sales, both domestic and foreign. The dominance of the PBF by European companies means this can only improve Europe's trade balance.

B.3.2.3 European Dimension

PyPy is an extremely high-profile project, as is only right since we intend to utilize the estimated 175,000 Python users as our user-base. Consequently, it is essential that we have the full support of the International Python community. Our success in that regard was more fully spoken about in B3.1, International Standards, but suffice to say that the only possible dimensions for this project is 'Europe Wide' or 'World Wide'. We could not find the expertise needed in a single country, and nor could we attain international acceptance without involving the top members of the Python community. Working at a smaller (single-country) scale would only 'fork the project' -- make a local version which is of limited use, while the main line project continued to develop in another direction. At best we would have a ghetto. At worst, we would have two hostile camps slinging insults at each other over which 'was the real Python'.

We have avoided all of these problems by carefully making the PyPy project sufficiently international, by discussing this proposal for nearly a year at international conferences, and by inviting non-European Python luminaries to our week-long intensive code-writing meetings called 'Sprints'. By producing a prototype in four coding Sprints (spread out over half a year) held in locations in Hildesheim, Germany; Gothenburg, Sweden; Louvain-La-Neuve, Belgium; and Berlin, Germany, while discussing the project on mailing lists and on our website, we have made certain that we have the proper dimension for our project to succeed, and become the new reference language for Python.

B.3.3 Innovation Related Activities

This is an Open Source Project. Thus complicated issues involving intellectual property do not arise. The knowledge produced by this project, every deliverable, is listed at Dissemination Level Public (PU). We are utterly committed to transparency and open dissemination, and as such will have all of our code available for nightly downloads, our papers freely available on the PyPy websites, and freely available for other people to link to. All Consortium members will have signed a consortium agreement, asserting that all code producing during the PyPy STREP will be released under the MIT Open Source License, approved by both the Open Source Initiative and the Free Software Foundation. The MIT license is here:

The MIT License

Copyright (c) <year> <copyright holders>

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

This license is concise and clearly permits any person obtaining our product to use it without limitation. The open, clear language removes any chance of a licensing controversy. It allows commercial entities and the project partners in particular to incorporate code and results into commercial or non-commercial projects.

The consortium did consider using the GPL license for protection from a commercial takeover. We recognise that there is some risk that the results of our project would be subject to an 'embrace and extend' strategy or another form of hostile takeover, especially if the project is a great success.

However, the 'critical mass' which makes such a scenario likely is dependent upon our achieving widespread adoption both within and outside the present Python community. Before we can get any adoption outside the Python community, we have to win the hearts of people on the inside. These people - especially the portal figures - have a strong preference for MIT license models and lack enthusiasm for the GPL.

Also, such a takeover attempt is not guaranteed to be successful and the Python world is probably more resilient to such tries than most other communities, since it is more strongly integrated and collaboratively focused than other comparable groups.

B.3.3.1 Management of Intellectual Property

Management of the primary PyPy intellectual property, source code, consists of posting the license on the website, or wherever source code is available, and periodically running a program, especially before software releases, to check that the license is properly referred to from each file.

The only other intellectual properties which we will produce are scientific papers, talks, workshops, and the like. They will all be freely available. Copyright shall rest in the authors, unless somebody gives a paper to one of the scientific journals that keeps all copyright to itself. In either case, no management is necessary.

The current version of Python is licensed under the Python Software License. Our project will in no way conflict with this license. This is authoritative. Tim Peters, a member of the PyPy project, is also director of the Python Software Foundation, and responsible for the Intellectual Property of the existing Python language.

B.3.4 Dissemination

To successfully disseminate knowledge from the PyPy project, the project needs to have good steadfast routines for documenting and interacting with the project stakeholders. The project management team (project manager and assisting project manager) will be responsible for overseeing dissemination tasks and activities.

Dissemination will consist of, but not be limited to

- The key activities in the project and the development process are the "Sprints". These are open forums to which we actively invite members from both commercial and research oriented organizations to actively participate
- since PyPy as a project has goal of a high transparency (see B5), documentation and information as well as project member contact information will be easy to locate for external interested potential stakeholders
- the PyPy project will gather on a regular basis, during Sprints, knowledge about the the development process and results. Every six months, the project will go through a project review workshop in which specific emphasis will be put on knowledge gathering and dissemination strategies
- a PyPy newsletter will be produced for external organizations/commercial enterprises that will be sent out after each Sprint, keeping them updated on project process and development progress. The assistant project manager will be responsible for this.
- to ensure that interested external parties will be able to use knowledge acquired in the PyPy project we will host two workshops during the project, one after 6 months and the first review workshop and one at the end of the project in which we actively work with interested parties to analyse potential usage of PyPy process and prototype. We will also host 4 domain specific workshops for SME:s within community, industrial stakeholders within embedded software community and game developing companies to ensure a thorough dissemination of possibilities of PyPy usage

Acronym: PyPy

We will also partake in the following official events, forum, conferences to spread information about the ongoing project, its unique process and technological impact and thus reaching out to software developers within the Python community as well as practitioners of other software languages:

EuroPython/EuroZope (European Python Conference), ACM/IFIP/USENIX International Middleware Conference, OSCON (Open Source Convention), OOPSLA (Object-Oriented programming, systems, languages and applications) PyCon (Python Developers conference), FOSDEM (Free and Open Source Developer European Meeting), ECOOP (European Conference for Object Oriented Programming)

More conferences will be added once the project gets started.

B.3.5 Exploitation

Our first goal in exploitation will be to make PyPy the reference Python language. Our market, then is all existing Python programmers. *Just how many are we?*

It is always difficult to measure how many people are using a programming language. Python is generally ranked the sixth most popular computer language in the world. Only Java, Visual Basic, C, C++, and Perl are believed to have more users.

The Python FAQ, available at the Python language home site of python.org says:

2.1. How many people are using Python?

```
Certainly thousands, and quite probably tens of thousands of users.
More are seeing the light each day. The comp.lang.python newsgroup is
very active, but overall there is no accurate estimate of the number
of subscribers or Python users. Jacek Artymiak has created a Python
Users Counter; you can see the current count by visiting
http://www.wszechnica.safenet.pl/cgi-bin/checkpythonuserscounter.py
```

Jacek's counter has more than 43,000 registered users.

Googling for 'python programming' gives 3,020,000 hits. ('Python' gives ten million, but many of those Pythons are probably actual reptiles.)

Compared to the most common languages these days, C, C++, Java and Visual Basic, there is certainly less 'market penetration', and in some niches, languages such as PHP, Perl and SQL are popular, but Python is vastly bigger than the more obscure languages such as Haskell, OCaml, Smalltalk, ADA, Ruby etc.

Some statistics from python.org may be relevant. There were 47,751 Python 2.3 downloads in the first 10 days of September 2003. These are the **bleeding edge** developers, who were interested in the new release of Python first made available at the end of August. At the other end of the spectrum, there are many more who wait for Python to become available as a Red Hat, or Debian package, or simply use the version of Python that came installed with their machine when they bought it.

For what it's worth, there are about 600,000 web visitors a month to python.org. About one third of them are using the documentation, which is a fairly good indicator that they are trying to use the language. Number of unique IP addresses per month is about 350,000 at this point, and has been rising steadily from about 250,000 this time last year.

Acronym: PyPy

Converting this into an actual estimate of number of users is difficult, because:

1. One user may use multiple machines (e.g. home & work & café)
2. Many sites use a firewall that hides the actual IP (and thus combines multiple users into one)
3. Not all Python users go to python.org every month (or at least not the Documentation page), because they have already downloaded the documentation for local browsing.

A conservative estimate would be that there are at least 175,000 active Python users in the world. At least half of them are in Europe, if O'Reilly's sales statistics for their popular books *Learning Python*, *Python in a Nutshell* and *The Python Cookbook* are to be considered relevant. Another piece of data, sales statistics of *The Essential Python* targeted at users of the Python version that compiles to the Java Virtual Machine indicate that there are at least 10 thousand Python users worldwide.

175,000. That's a lot of Python users. Greatly improving the language which they all use will have an enormous impact. Since the PyPy development team is in constant contact with the Python community, and its world-wide leadership, there is no friction or political resistance to PyPy. The individual members of the consortium are among the most well-known and well-respected members of the Python community. We have taken special care to include prominent stakeholders in the existing language from the very beginning, and have constantly invited members of the community to a series of development Sprints whereby we produced a prototype which served as a proof-of-concept. We have also budgeted for extensive communication and dissemination with this community. Thus when we deliver a working product, they will be ready to adopt it.

Beyond this, there are the users of other languages. There are something like 12 million programmers world-wide and roughly 50% of those use Visual Basic (according to International Data Corp). In March 2002, Borland said Java had about 1.5 million developers. How do we go about getting them to use PyPy?

First of all, we must give them something that they find useful, and useful enough to switch. While Python users might be satisfied at getting a faster language, the users of other languages, who will need to make a greater effort in switching need a greater reason to switch. Fortunately, this is the technical goal of the project.

The current State-of-the-Art, both in Python and in programming languages in general, does not best serve the new needs of the creators of embedded, networked and distributed software sectors. They need a more flexible language which is easier to reduce to its 'bare-bones' for embedding, and which can dynamically reconfigure itself and optimize execution speed at the interpreter level. Consequently, the PyPy project is a collaboration between academic researchers and SME software and service providers. The former have the skill and vision to produce a new run-time language architecture for the twenty-first century and the latter wish to deploy PyPy in their innovative new business ventures.

Furthermore, an important aspect of the current proposal is that we will make use of the new opportunities that the Object Spaces give us to extend the reach of the Python to areas where programmers currently have to resort to other languages. In particular, we will bring the Stackless version of Python into the mainstream. This allows Python to handle thousands of threads inexpensively, making simulation software easy to program. The makers of computer games have already shown interest in this, and have funded its early development. The Stackless Python implementation also supports mobility of running code, allowing it to migrate from one machine to another. Other programming areas addressed by the project by building on Object Spaces are Aspect Oriented Programming, Design by Contract, and Constraint Solving. All these will significantly increase the attractiveness of Python as a programming language, and we will work with people in

Acronym: PyPy

the Python community who are interested in simulation, software engineering, and the semantic web to ensure that they can take full advantage of the new platform we are offering.

Here is a selection of endorsements of the project from stakeholders, members of the Python Business Forum.

Kaval Wireless Technologies Inc. uses Python in a variety of ways in the development and production of its wireless coverage extension systems, as well as using it in some of the products themselves. The PyPy project holds great promise for improving the performance and cross-platform nature of the Python environment, and I look forward to seeing those improvements occur.

(Peter Hansen, P.Eng. Director, Software Engineering, Kaval Wireless Technologies Inc.)

I'm interested in PyPy because it's about Python, the pragmatic language that I use for my business and make my living with, and because it promises pragmatic benefits (performance is my personal #1, but there is a lot of other potential).

(Martijn Faassen, Infrae Netherlands)

I am one of the main developers of the open source Twisted networking framework, which is used by organizations ranging from NASA and California state agencies to individuals and companies in the US, Italy, Sweden, France and the Netherlands. I would like to see PyPy succeed as the result would be a platform that is far superior to competing software platforms, while still being open and not tied to the fortunes of one or two American corporations.

(Itamar Shtull-Trauring, Chief Technology Architect, Zoteca)

I'm an independent software developer and consultant. My current focus is the development of design tools, code generators, and monitoring tools for embedded, distributed, real-time systems (often applied in safety-critical domains). I have been using Python with great success for these application domains.

A successful PyPy would increase the benefits of using Python:

- *it would allow me to even more functionality into Python and further reduce the amount of C code needed for the on-line part of such systems (which is running on PC-class HW).*
- *it might allow to replace (some but not nearly all) C code in the embedded parts of such systems.*
- *it might substantially improve the performance of the design tools and code generators.*

(Christian Tanzer)

As a vendor of development tools for Python programmers, we support PyPy as an important step towards advancing the technology behind Python programming language.

(Stephan Deibel, CEO Archaeopteryx Software, Inc.)

Acronym: PyPy

The results of the PyPy project are also expected to be utilized by established industrial users. We have expressions of interest from *Bang and Olufsen* , the Danish manufacturer of high-end stereo equipment, *The IBM Zurich Research Lab* , *Vodaphone* and *Ericsson* the mobile industrialists, *Siemens* , the German conglomerate, and *Axis* the Swedish-based multinational market leader in in-house developed chip technology for network video and print servers.

References:

[PEP1] <http://www.python.org/peps/pep-0001.html>

B.4 The Consortium and Project Resources

B.4.1 Description of the Partners

B.4.1.1 DFKI

Role: Project Coordinator & Technical Partner
Country: Germany
Contact: Alastair Burt

Founded in 1988, the DFKI (<http://www.dfki.de/>) today is a contract research institute in the field of innovative software technology based on Artificial Intelligence (AI). The DFKI focuses on the complete cycle of innovation - from world-class basic research and technology development through leading-edge demonstrator and prototypes to product functions and commercialization.

Based in Kaiserslautern and Saarbrücken, the German Research Center for Artificial Intelligence ranks among the important "Centers of Excellence" worldwide.

An important element of DFKI's mission is to move innovations as quickly as possible from the lab into the marketplace. A tenet of the DFKI is that the best way to meet its technology transfer goals is to maintain a varied portfolio of research projects at the forefront of science.

B.4.1.1.1 Key Personnel

Alastair Burt, a researcher at the German Research Center for Artificial Intelligence (German Research Centre for Artificial Intelligence). He studied Psychology at the University of Stirling, Scotland and Computing at Imperial College London. In 1990 he joined the Multiagent System Research Group at German Research Center for Artificial Intelligence GmbH in Saarbrücken in the department of Deduction and Multiagent Systems headed by Prof. H. J. Siekmann. Since then he has been involved with a wide variety of research projects centred around the application of multi agent systems, including participation in several EC projects and taking responsibility for their coordination. In particular, he was coordinator of the ASWAD project that developed a Free Software workflow tool for public administrations across Europe, an innovative use of the Zope and Python platform.

B.4.1.2 University of Southampton

Role: Technical Partner
Country: United Kingdom
Contact: Michael Leuschel

The University of Southampton (<http://www.soton.ac.uk>) is one of the leading universities in the UK for high-quality research across a wide range of disciplines.

In the most recent Research Assessment Exercise (RAE) carried out by the Higher Education Funding Council for England the University achieved spectacular results - gaining the 5 or 5* status for 24 of its 34 units of assessment (5* being the top rating). The University was placed ninth in the country for the quality of its research. The University of Southampton competes internationally as an elite research intensive institution.

B.4.1.2.1 Key Personnel

Armin Rigo was born in 1976 in Lausanne, Switzerland. He is a researcher at the University of Southampton (UK). He studied Mathematics at the University of Lausanne and obtained his Ph.D. in Logic and Set Theory at the Free University of Brussels. He is the main author of several commercial, open source and research programs and contributed to a number of them, most notably in the fields of computer graphics and 3D modelling, education, and programming languages. He recently developed in the Psyco project novel techniques for efficient interpretation of dynamic programming languages. He is also a member and contributor of the TUNES Project for a Free Reflective Computing System.

Michael Hudson was born in 1978 in the United Kingdom. He holds a first class degree in Mathematics from the University of Cambridge and took the Certificate of Advanced Study in Mathematics at the same institution. He is currently studying for a PhD in Algebra at the University of Bristol, but should join USH upon graduation. He has been a member of the wider Python community since 1998 and has had commit rights to the Python core since August 2001. He was release manager for the 2.2.1 release and made many contributions of code over the years. He is the chairman of the EuroPython Society and is a member of the Python Software Foundation (PSF).

B.4.1.3 AB Strakt

Role: Project Management & Technical Partner
Country: Sweden
Contact: Jacob Hallén

AB Strakt (<http://www.strakt.com>) is a software product developing company located in Göteborg, Sweden. The main product of the company is CAPS, a platform for constructing collaborative workflow systems. On top of this platform the company has built a number of applications; helpdesk, project management, customer relations management and procurement. All products are built in Python. Strakt is involved in the development of several OpenSource projects, which in one way or another play a role in the company's software development. The main development platform is Linux. The company has 13 full time employees, and 6 part time employees. The company was formed in January 2001.

B.4.1.3.1 Key Personnel

Jacob Hallén, born 1958, comes to his position as co-founder and CTO of AB Strakt from being a Technical Manager and international standards expert at the LIBRIS Department of the Royal Library. He was the LIBRIS representative in the EU funded ONE-2 project. Before this, he was the CEO of NetGuide Scandinavia AB, one of the first internet services companies in Sweden. Mr Hallén has also been a Computer Science and Programming teacher and an army officer. Throughout his career he has managed many projects which varied from 1200 participant conventions down to 3 person development tasks. He has also done electronics development and microcontroller programming, winning two innovation awards in the process. Mr Hallén is also chairman of the Python Business Forum.

Tim Peters : Over 20 years top-tier industrial experience in programming language implementation and high-performance computing. 1979-1988, Cray Research: Compiler development, Group leader--common back-end optimisation group. 1988- 1994, Kendall Square Research (KSR), Compiler and Library development, Architecture and FPU design. 1994-2000, Dragon Systems:

Acronym: PyPy

Developed core speech recognition system for portable devices; scalable, large-scale telephone speech recognition; and award-winning PhoneQuery Toolkit product. 2000 to present: Zope Corporation: Development--Core technologies underlying Zope's leading content management framework; Python core. Python: first port of Python to 64-bit platform (KSR-1); POSIX pthreads support; algorithmic and optimisation expertise; elected Director of Python Software Foundation (PSF) since its inception.

Samuele Pedroni : Born 1974 in Switzerland. Dipl. Math. ETH Zurich (1999) His thesis was awarded by the ETH Polya Fond. Between 1999-2001 he worked as teaching/published research assistant at the Institute of Theor. CS of the ETHZ. He was designer on the state-of-the-art genetic programming framework for Java JRGP, becoming involved in Jython (the industry-strength Java re-implementation of Python). He is now a main developer of Jython, working on internals, compilers and Java integration, and was author of Jython Essentials (O'Reilly, 2002). He is also involved on the ongoing design of Python. For his contributions to Python/Jython he was nominated and accepted as a member of the Python Software Foundation. He brings to the project his know-how on languages, re-implementation/design of Python, reflection, lookup and dispatch optimisation.

Laura Creighton : Co-founder and lead investor of AB Strakt, Treasurer of the PBF. Studied Physics (csc minor) at the University of Toronto, and later instructed there in Physics and Computer Science, while simultaneously working for the Canadian Armed Forces, teaching programming to non-programmers, and assisting with research in Human Factors Engineering and Learning Techniques. Moving to the US, she consulted for software companies and government institutions, taught Unix, project management, and interpersonal relations, and wrote a geophysical simulation system. She brings strong connections in financial and government sectors, and was an Open Source advocate before the term was coined. Her passions: programming, empowerment of ordinary citizens, and the Open Society.

B.4.1.4 Logilab

Role: Technical Partner
Country: France
Contact: Nicolas Chauvat

Logilab (<http://www.logilab.fr/>) specializes in the use of Python for advanced computing, artificial intelligence and knowledge manipulation. Logilab works for large public and private entities in France and Europe, including Commissariat à l'Energie Atomique, Electricité De France, SNECMA, etc. Logilab was involved in two IST projects before, ASWAD (free software workflows for public administration) and KIDDANET (web filtering for kids based on machine learning techniques). The company has currently seven full-time employees and is expected to reach the size of ten by the end of 2003.

For over three years, Logilab has been dedicating resources to high-profile projects such as intelligent agents, natural language processing and semantic web applications. Logilab has also been contributing libraries to the Python languages, namely XML processing, logic and aspect-oriented programming and static checking. Since Logilab has been committed to free software from its creation in 2000, most of these projects are available under a free software license from the Logilab.org website.

B.4.1.4.1 Key Personnel

Nicolas Chauvat : He completed his studies of engineering (spec. Robotics) at Ecole Centrale Paris and computer science (Artificial Intelligence) at Université Paris 6. Worked as a researcher in industrial and academic laboratories in France and in the USA before founding Logilab in year 2000. Research work concerned the modelisation of complex distributed electronic systems as agent communities and human-agent interaction based on context-awareness capabilities. President and CEO of Logilab. Python user since 1997. Secretary of the Python Business Forum. Chairman of the "EuroPython Science Track".

B.4.1.5 ChangeMaker

Role: Project Management
Country: Sweden
Contact: Beatrice Düring

Change Maker (<http://www.changemaker.nu>) is an education and consultancy firm and we offer services in the areas of project management, leadership, team building and change management. We supply courses, workshops and seminars to corporations, schools and other organisations.

We offer support for applications and process management to small companies in receiving financial support from the European Union program Växtkraft Mål 3.

We also tailor educational concepts for the national Qualified Education committee (KY), aiming at making the process of evaluating the needs for recruiting personnel easier. Examples are companies working with interactive media and games development.

We deliver project management and quality evaluation for larger educational projects.

Some customers:

Blekinge Tekniska Högskola, Socialhögskolan, Arvika Näringslivscentrum, Elmo Leather, Learning Tree International, FSO, Semcon, Lundsbergs Internatskola, Galaxenis

B.4.1.5.1 Key Personnel

Beatrice Düring studied teaching/pedagogy at the University of Karlstad in Sweden. She was recruited into the IT-industry to work as a project manager for large scale education projects for the company NetGuide Scandinavia, Gothenburg. Since 1998 she has been working with education and development project management and management of education and consultant departments, implementing Open Source strategies and Agile development methods. Beatrice also teaches project management, leadership and communication courses for Learning Tree International.

B.4.2 Foreseen Amendments in Consortium Structure

In the original plan for the consortium structure, the Python Business Forum (PBF), an international non-profit organisation founded to promote the use of Python in the business world, was intended to fulfill two roles:

1. to act as a conduit for the funding of the sprint programming workshops, and
2. to distribute technical work, in a flexible manner to a pool of freelancers.

During negotiations the Commission indicated that the current organisational structure of the PBF was not particularly suited for participation in IST projects. Other ways have been sought to fulfill the two roles in a different manner:

1. It is intended that some of the experts who participate in the sprint workshops will fill out the paperwork necessary to become members of the consortium with the status of *physical persons*. Currently, the exact list of such persons is not known. As they become known, they will be added to the project consortium through an amendment to the contract. In the initial contract, the budget for these physical persons is assigned to the University of Southampton.
2. During the negotiation phase, the three major technical experts, Holger Krekel, Christian Tismer, and Alex Martelli, who were intending to participate in the project through the PBF were in the process of forming limited companies in order to participate in the project as full partners. The negotiation schedule did not allow the newly formed companies to be part of the initial consortium, so their full participation will also necessitate an amendment to the contract. The tasks that these three future partners are intended to carry out are assigned to the DFKI in the initial contract.

B.4.2.1 Foreseen Future Partners

B.4.2.1.1 Holger Krekel : born 1969 in Frankfurt a.M., began in 1985 to work as a lead programmer producing games for Electronic Arts. He went to university, gave courses in Prolog, C, Assembler, mathematics and assisted in numerical computing. He got his degree "magna cum laude". He consulted for Volkswagen, large German banks and the chairman of the EU-founded CEN/ISSS workshop who contracted him to prototype integration of OpenSource software. He implemented an open-source transaction service on top of TAO/CORBA and published several articles about Free projects. In 2001 he joined the Python community, published an interactive tool, took part in Zope3 development and some of the first coding Sprints in Europe and became one of the initiators of the PyPy project.

Expected contribution to the project: 24 man months.

Acronym: PyPy

B.4.2.2 Alex Martelli : Best-selling author of *Python in a Nutshell* . Co-editor of *Python Cookbook* . ActiveState 2002 *Activators' Choice* award winner. PSF member, Python language developer, PBF board member. Consultant to AB Strakt, developing the CAPS framework. Also consults for other firms on Python and O-O design, teaching, coding, feasibility studies, interfacing. 1989-2002, Cad.Lab (think3, Inc): innovative component architecture for web-enabling existing GUI- oriented apps; Event Manager, interfacing, proprietary protocols. Taught Computer Programming and Numerical Analysis, Ferrara University. 1981-1989, IBM Research: 3 Outstanding Technical Achievement awards, voice recognition, image processing. *Laurea* 1980, Electronic Engineering, Bologna University, 100/100 magna cum laude.

Expected Contribution to the project; 12 man months.

Christian Tismer : Born 1956 in Jena, Germany, studied Math, Physics and Informatics at the Free University of Berlin, diploma on Adaptive Huffman Coding. He has been working for Pharmaceutic Research companies for more than 10 years, doing statistical evaluations, EEG and EMG recording, signal analysis, networking, report generation and automation. He wrote his first multitasking system in 1985 for DOS, continuous 32 channel EEG recording and visualisation on a 286 machine. Later, he worked on Netscape plugins, document automation, Web and database applications. Converted to Python in 1997, founded Python Starship, became Python core developer. He translates Python books into German, is the author of the Stackless Python extension to be merged into PyPy, and is one of the founders of the PyPy project.

Expected contribution to the project: 24 man months.

B.4.3 Sub-Contracting

Some pure accounting and auditing tasks will be subcontracted. Some documentation and tutoring tasks may be subcontracted. No core technical functions or project management will be handed to non-partners.

B.4.4 Other Countries

While this project is of international interest, no parties outside the European Union and its candidate countries will take part in EU funded activities..

B.4.5 Quality of Partnership, Involvement of Users and SMEs

B.4.5.1 Roles

While all the partners except ChangeMaker have staff with Python programming skills that will enable them to fulfill their responsibilities in the various tasks of the project, each partner brings unique skills or functions, without which the project is not complete.

DFKI has previous experience of being a project coordinator in EU projects, ensuring smooth communication between the project and the FP6 project officer.

Acronym: PyPy

University of Southampton is the employer of Armin Rigo, who is the lead architect of the whole project as well as the author of Psyco, the blueprint for how to do optimisation in PyPy.

Strakt brings management know how and entrepreneurial skills to the project as well as showcasing how to apply the results of the project in a major business application.

Strakt will also bring Tim Peters and Samuele Pedroni to the project. Tim Peters been a core developer of the Python language, what we call CPython, from its inception. Apart from being an outstanding programmer, Tim has intimate knowledge about all the details of the Python language definition. As Director of the Python Software foundation, Tim Peters is also responsible for the Intellectual Property of the existing Python language.

Samuele Pedroni is currently the main developer of Jython (the industry-strength Java re-implementation of Python), working on internals, compilers and Java integration. A member of the Python Software Foundation, he brings to the project his know-how on languages, re-implementation/design of Python, reflection, lookup and dispatch optimisation.

With 13 full time and 6 part time employees, Strakt is an SME representative. Strakt is particularly interested in this project because it is doubtful that its Framework system, CAPS, can scale from several hundred thousand to several million concurrent users unless one of two things happen. Either parts of CAPS are rewritten in C, which is faster, or Python itself becomes faster. The second alternative is much to be preferred. Participation will also enhance Strakt's ability to attract the best Python programmers, and add to it's reputation in the Open Source community. Strakt's financial backer and board of Directors enthusiastically endorse participation in this project as a brilliant strategic move.

ChangeMaker adds rare project management skills by managing not only the project, but the learning processes of the project participants as well as the group dynamics between the different members. We intend to document and disseminate the management of change throughout the project. ChangeMaker is also the contact point for Axis Communications, who will receive a specific report on how to integrate PyPy in an embedded device.

Logilab focuses on constraints and aspect oriented programming and will verify that PyPy is both extensible with specialised language features and embeddable in small-sized devices with dedicated hardware.

With 7 full time employees, Logilab is an SME representative.

Holger Krekel will have a focus on development, packaging and dissemination tools. He will also be a main contributor in matters of systems architecture.

Christian Tismer is the developer of Stackless Python, which is the blueprint for how we intend to implement persistent threads.

Alex Martelli is a prolific writer and popular speaker as well as a Python programmer. Alex is uniquely suited for widely disseminating the progress and the results of the project.

B.4.5.2 Ability to Deliver

DFKI, Logilab, and University of Southampton have already successfully participated in EU projects, so their ability to deliver on another project should not be in doubt.

Strakt, Tismer and Martelli have all recently produced substantial products, which show their capability to handle large undertakings.

Acronym: PyPy

Krekel has demonstrated his ability to deliver results during the prototype phase that has preceded this application.

ChangeMaker has a number of documented successful projects. Further details can be found under B 5.1.

A special circumstance is that several of the project participants have already collaborated in developing a proof of concept for PyPy. This means that the ability to collaborate and to deliver results has already been tested under circumstances that are very similar to what they will be during the project.

Moreover, this project will be run in as transparent a fashion as possible. Our code will be available every night from our repository. Our mailing list discussions will be available on a publically readable archive, and any person is free to join our mailing lists. Our Internet-Relay-Chat conversations, when important, will be logged and archived. You can watch us create our documents, our code, and our deliverables, day to day, as it happens. Meetings in person will be summarised, and they will be posted. This is *life in a fish bowl*, as transparent a process as you can see.

This is important because the reason most projects fail to deliver is because somebody was having a problem, and was too embarrassed or afraid to look bad in front of others to admit the problem until it was too late to do anything about it. It is secrecy, and not lack of competence that causes most of the problem. We will avoid this. Our EU project leader will be able to monitor our progress on a daily basis and will always be able to know what we are up to, and how we think it is going. He or she will be free to participate in our Sprints, and join our mailing lists, talk to us via IRC - whatever level of involvement is desired.

B.4.6 Resources to be Mobilised

In any project there are three crucial resources to be mobilised.

The first is equipment. The PyPy project needs very little. We would like to purchase a projector, for use in displaying code at Sprints. We would also like to purchase a portable printable whiteboard. We draw a lot at Sprints, and would like to be able to give every attendee a copy of the diagrams we made at the touch of a button for use at home in between Sprints. Each of these will cost somewhere between 1 and 2 thousand euros.

The second is Finance. We don't need to mobilise outside financial contributions, though we have some excellent connections. AB Strakt, Krekel, Martelli and Tismer have already arranged for bank guarantees, should they be required by the Commission. The PBF and AB Strakt both use KPMG as their standard auditor, and a KPMG recommended bookkeeper who is familiar with EU project funding for their daily business practices. Everybody else has already been involved in successful EU projects, and will simply continue their usual behaviour.

And the third, and most important resource to be mobilised, is people. This is where this Consortium really shines. Not only have we already attracted some of the top people in the Python community in this project, we also have a plan to continue to attract the interest, support, and constructive criticism of the very best. The Open Source movement is **all about community**. This fact is often overlooked by people who are discovering Open Source for the first time, who mistakenly believe that it is **all about licensing**. Successful Open Source projects are based on

Acronym: PyPy

sharing and trust, process rather than product, and key people. The same is true of Agile development methods, as defined by the Agile Alliance.

We are pioneers of a software development method, Sprint Driven Development, which promises to mobilise people in a unique and special way. We outline *what* we do in Section 5 -- Project Management. Here we would only like to speak briefly of *one reason why we do this* .

One of the classics of software management is Fred Brooks *The Mythical Man Month* .

In it he asserts the then-controversial, and now well-established claim: **Adding people to a late software project makes it later** . This is because the communications cost of *bringing a new person up to speed* outweighs the benefits you can get by putting them on the project. And some projects are just too large -- the burden of simply letting the left hand know what the right hand is doing is crippling.

Sprint Driven Development is an attempt to make a development model which refutes this claim. What would you get if you didn't have to *bring people up to speed* because *they were already aware of the project* ? If you published everything you were doing and made constant efforts to communicate where you were, and why you were doing things to the base of programmers you might *someday* like to have on your project, then when time came to add people to the late project (or just because the project reached a point where there *was* comparatively more work that could be shared among programmers) they would already be mostly ready to go.

You could not do that in a proprietary environment. You **can** do this in the Open Source Software community. A for-profit company cannot afford to have very many people just sitting around, preparing, absorbing knowledge, in case they might be needed somewhere. Commercial software companies pay their people to write code. But outside the company is the competition. Sharing is thus frowned upon.

But there are three groups who function this way, all the time. They are always sitting around learning things. The first group are students, and teachers. The second are those people in industries which have an off-season -- tax preparers and fishermen both fit this pattern. And the third are software consultants, both small and those in large IT consulting firms. Learning things is called *keeping current* and is essential to maintaining competitiveness as a software consultant. I will call these people **seasonal workers** though the term is not quite accurate. The important thing is that you can mobilise them for short projects. And the Open Source community *has* mobilised them quite effectively.

Sprinting is more than just an effort to maximise creativity by putting all your creative people in a room and letting them bounce ideas off each other and write code to test theories. (Though it is that, too, and that is very important.) Sprinting is also an way to constantly involve the community and disseminate knowledge, especially the more difficult **know-how** , as opposed to **know-that** which one can read in books, papers, mailing lists and websites. It keeps people *in the loop* -- ready to contribute should it be necessary -- especially when combined with the sort of supplementary materials we intend to produce. (**Know-that** is obviously good as well, but, it is **know-how** that is must be developed when bringing a new person up to speed in a project.)

As part of this project, we intend to document our creative process in the hope that it will be picked up throughout the Open Source community as an effective way to work, and one that is compatible with the EU's Funding Programmes. Right now there is considerable desire on the part of Open Source developers to become involved with EU projects, but too little experience. The process seems cumbersome, and overweildy for those people who are used to more Agile methods. There is great interest in finding an acceptable way to respond to calls and provide enough accountability to

Acronym: PyPy

suit the EC, while enough flexibility to suit the Agile developers. We believe we have found such a way, and are very anxious to prove it so, and then spread the message so that other Agile developers can benefit from our experience.

B.4.7 STREP Project Effort Form

Full duration of project: 24 months

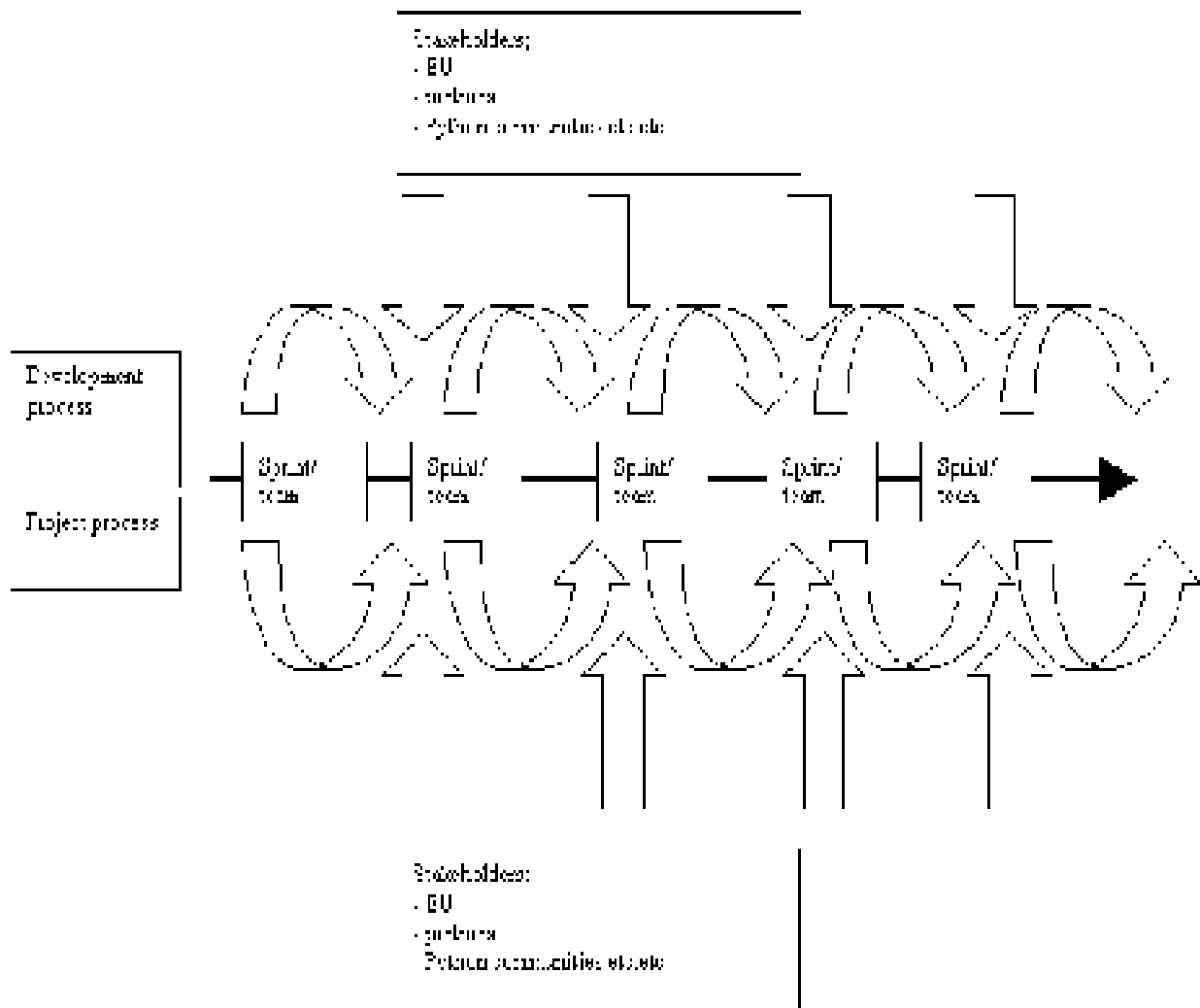
	DFKI	USH	Strakt	Logilab	CM	Total
Research and Innovation						
WP2: Infrastructure and Tools	8					8
WP3: Synchronisation with Standard Python	6	3				9
WP4: PyPy Core	12		6	5		23
WP5: PyPy Translation	6	8	15			29
WP6: Core Optimisations	1	12				13
WP7: Translator Optimisations	15	3				18
WP8: Dynamic Optimisations	2	6	17			25
WP9: Search and Logic	10			9		19
WP10: Aspects and Contracts	3			9		12
WP11: Specialised Hardware				5	1	6
WP12: Security, Distribution and Persistence		3	10			13
WP13: Integration and Configuration	8			4		12
WP14: Documentaion and Dissemination	12				11	23
Total Research and Innovation	83	35	48	32	12	210
Management						
WP1: Coordination and Management	4		4		4	12
Total	87	35	52	32	16	222

B.5 Project Management

PyPy as a project will be implementing an agile development lifecycle. This choice of development method will have effects on the way the project will be structured and managed.

The project will have a structured project plan as is showed in this proposal (workpackages, Gantt-chart, deliverables, quality control etc). It also means that the project process, once the project get started, will work from a evaluate-feedback-change perspective, or so called "learning loops" in which project management will continuously follow up on the initial project plan but also evaluate process, team effectiveness, communication climate. From these learning loops change, when necessary, will be applied throughout the process.

To illustrate the focus on development process as well as project focus:



Both the project and the development process are based around critical workshops, so called "sprints" that will take place on a six week cycle throughout the project (24 months). Around these sprints input and output to stakeholders will be structured. The arrows above symbolize the evaluation- feedback-change system that will be implemented.

Acronym: PyPy

This method will affect the role of the project management, management structure, role of coordinator, project meetings, quality control and communication in the project in what we have experienced to be a very constructive way.

Our reasons for choosing this development and project method are several:

- This project has a history of 6 months in which the team successfully implemented sprints and agile development methods
- In this project, team members from at least 5 different countries will work continuously in separate places. Sprints will be the main forum in which the team members meet up and work together in real life
- The sprints will be open for non team members to participate in the development process, thus allowing for an open and feedback driven process
- The sprints will be the forum in which knowledge will be shared and the transparency within the project organisation will be measured

We will during the project focus on evaluating and documenting our project method and share knowledge and experience in that area as well. It is our goal that the overall deliverables from this project will be a functioning PyPy as well as an effective project method for agile development/Open Source projects. Our goal is also to disseminate this knowledge to the developer communities inside and outside the Open Source movement as well as to commercial and academic organisations.

On the following pages we will describe in more detail how this choice of method will influence the way this project will be managed.

B.5.1 Coordination and Management Team

The PyPy project will have a management and coordination structure that is based upon three resources, Jacob Hallén as project manager, Beatrice Düring as assisting project manager, and Alastair Burt as coordinator.

The role of the project manager is to:

- manage the project and its scope of time, budget and deliverables
- lead the work of the management board and report to the management board
- execute decisions made in the management board
- report to the project coordinator
- support the project coordinator concerning the relations to the EU
- manage the sprints
- manage tracking of quality assurance of the technical development

The role of the assistant project manager is to:

- report to the project manager
- participate in reports to management board and project coordinator
- manage project administration (reports, documentation, etc)
- manage routines and tracking of sprints, quality assurance of project process, resource allocation
- manage contact with external partners
- manage the day-to-day operations of the project (ex. executing decisions made by management board)

Acronym: PyPy

- manage the knowledge process and actively spread information to the
- stakeholders regarding methods used and knowledge acquired

The reasons for having a structure based on a project manager and assisting project manager are:

both the development and the project process will receive due attention in that the persons chosen have expert skills in these different areas

the project will not be exposed to the risk that a single project manager would mean

a project of this size with team and stakeholders distributed in several countries needs more project management resources

The responsibilities of the project coordinator will include:

- to manage negotiations regarding proposal
- to manage ongoing dialogue with the EU project office during the project
- to appoint a project manager
- to be a representative on the management board
- to participate in the project review workshops
- to submit reports regarding project review workshops
- to participate in project evaluation and submit report of this to the EU project office
- handle financial tasks and payments between the project and the EU project office

The project coordinator will be able to use the project management team for support in the tasks mentioned above.

The skills and experience of the combined management and coordination team are as follows:

B.5.1.1 Large Scale Projects

Jacob Hallén has been working since 1994 with large scale development projects. He was a consultant for, and later employee of, the LIBRIS Department of the Royal Library of Sweden (<http://www.libris.kb.se>) in the role of Technical Project Manager, with main focus on being systems architect for the national bibliography system and interlibrary loan system. Participated in international standardisation groups for search systems (Z39.50) and interlibrary loans.

He was also the initiator of the international standardisation effort for library services information. Participated as the Royal Library representative in ONE-2, an EU funded project under the Telematics for Libraries project (<http://www.one-2.org>)

Since 2001, Jacob has been involved in founding AB Strakt (<http://www.strakt.com>), a company developing workflow and document handling systems. There he has worked in roles as developer, project manager, CTO and CEO. The company has grown from 3 employees to having 13 full time employees and 6 part time employees so far.

Connected to his work at AB Strakt, Jacob has also been active as co-founder and chairman of the Python Business Forum, an international trade organisation for companies that use Python as their main tool of business. The PBF has approximately 50 member organisations. He is also the project leader for the EuroPython 2004 conference, to be held in Göteborg, Sweden 9-11 June 2004.

Beatrice Düring has experience in large scale education projects involving working with consortia of three companies servicing a stakeholder group of about 30 recruiting companies. These large

Acronym: PyPy

education projects was part of a national program to solve shortages of skilled IT-personnel during the years 1998- 2000. 200 students participated in the projects and the projects met their deliverables in that over 80% of the student were employed after the education. The project team consisted of 7 persons working full time. As a project manager, Beatrice was responsible for meeting project goals, meeting profit margins, leading the team and creating strategies for stakeholder participation in the projects. She was also responsible for reporting and documenting the project to the client.

Since 2000 she has been involved in similar assignments, one recently finished for University of Blekinge in which the education was directed towards recruiting companies in the game development industry. She has also worked as project manager for several development projects during the time 1998-2002.

She has also developed project methods for the companies and teams shes been working with and have also been working with quality assurance of development projects. Her current company, Change Maker is also working with supporting smaller companies in the application process for the EU Framework 3 (Växtkraft Mål 3) and has a experience of working with similar EU-funded projects since 1997.

B.5.1.2 Financial Tracking in Projects

Jacob Hallén has a widespread experience of founding and managing companies as well as being project manager for large scale projects. He has also developed several accounting programs. When being the CEO of NetGuide Scandinavia AB, the company was under budgetary squeeze in its early days, generating a lot of experience in tight cost control and progress tracking. The management was successful and the company grew to 35 employees under his leadership.

The large scale education projects that **Beatrice** managed had a profit margin of 20% which was met. The total budget for these projects was SEK 20 million. She has also recently been involved in the prestudy, budgeting and start of a 6 year long education project in Arvika, Sweden with a total budget of 18 million SEK.

During her time as a manager for the education and consultant department in NetGuide Scandinavia (1999-2002) she had budget and result responsibility.

B.5.1.3 Leadership Skills

Jacob Hallén has experienced leadership challenges in different situations. In his role as an officer in the reserve of the Swedish army he has been deputy rifle platoon leader in the Swedish UN forces in Cyprus, duty officer with responsibility for the battalion safety and security in Lebanon and instructor/platoon leader for training raw recruits. He has been a teacher at the Chalmers University of Technology, for Informator and for LearningTree International, all of which include being a leader for your students. At NetGuide Scandinavia his leadership was mostly focused on leading the company, initially with 4 people. A number that grew to 35 in the subsequent 3 years.

At LIBRIS he assisted the project leader for the SEK 20 million modernisation project in getting consensus among the approximately 30 members of the consortium on what sort of changes should be required, wanted or tolerated in the new system.

Acronym: PyPy

At AB Strakt, Jacob Hallén started out managing the company but changed his role to Chief Technical Officer, after successfully recruiting a suitable CEO as replacement. Jacob enjoys managing technical processes more than general corporate management.

Beatrice Düring has experience from leadership situations in projects as well as in line organisations since 1998. During four years she was a part of a management team of five people, leading teams of 5 to 14 people. As a leader, Beatrice was responsible for coaching, motivating and developing her personnel.

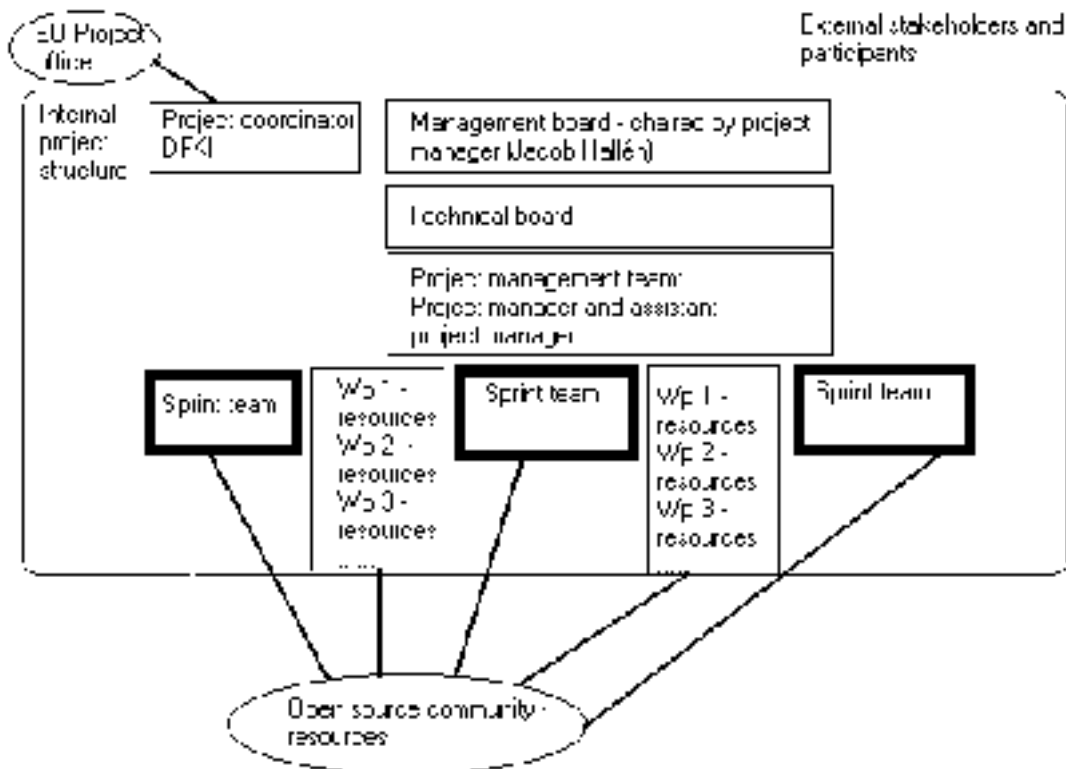
Beatrice employs strategies of empowerment, active listening combined with creating and maintaining an open communication climate based on honesty and trust to achieve goals together with her team. Beatrice has been teaching management oriented courses (leadership, project management, communication, conflict resolving) for Learning Tree International since 2000 in both Sweden and USA.

B.5.1.4 Coordination in EU Projects

Alastair Burt, DFKI, has participated in several European projects, was the coordinator of the EU-funded ASWAD free software project, and is in the coordinating team of the EUTIST-AMI cluster. The DFKI itself has over a decade of experience of working with the EU's financial and reporting procedures.

B.5.2 Management Structure

The management structure will be as follows:



Acronym: PyPy

Representatives in the management board:

- Alastair Burt, DFKI
- Holger Krekel
- Jacob Hallén, AB Strakt
- Nicolas Chauvat, Logilab

Responsibilities of the management board includes:

- manage resources
- manage quality assurance for the project and the technical process
- track cost, timeline, tasks, budget
- manage technical implementation plan and progress
- manage compliance with legal and ethical obligations

The operative management of these responsibilities will be delegated to the project management team and the technical board. In the management board, decisions that cannot be made by consensus and open discussions will be solved by voting (each partner have a vote, in case of tie - the project manager will have a casting vote)

Representatives in the technical board:

- Armin Rigo, University of Southampton
- Samuele Pedroni, AB Strakt
- Holger Krekel
- Christian Tismer

Further representatives on the Technical Board will be selected at the outset of the project. They will be appointed by a vote of everyone who has commit rights to the source repository and used them to contribute source code. Guido van Rossum, the author of Python will act as an advisor to the Technical Board.

The responsibilities of the technical board are:

- make recommendations on technical strategic issues to be decided in the management board
- manage the quality assurance of the technical development process - this task is delegated to the technical board from the management board
- manage technical implementation plan and progress - this task is delegated to the technical board from the management board

These responsibilities are reported to the management board through the technical director who is a representative on the management board. In the technical board, decisions that cannot be made by consensus and open discussions will be solved by voting (each representative have a vote, in case of tie - the technical director will have a casting vote)

Since the PyPy project is implementing agile/Open Source methods ("Sprints") the goal is to have a proactive team of developers and project managers. This means that the project will be ruled by one primary management strategy - to delegate as much responsibility to the developer team and the persons responsible for each individual workpackage. This rule `_will_` guide all planning and decision-making in the two boards.

B.5.3 Project Meetings

Management Board will meet at the start of the project and two times per year or on an ad hoc basis as requested. The meetings will normally be scheduled to rotate between countries of the EU and mainly the principal contractors home base. The project manager is responsible for the invite and agenda as well as managing the meetings. Objectives on these meeting are tracking progress regarding workpackages, budget, timescale and strategies for involving stakeholders as in partners or new partners. Agenda and discussions/decisions on these meeting will be documented and put up in the internal project web.

B.5.3.1 Team Meetings

The project team will meet at the "sprints" which take place on a six week cycle (see below). During the sprints, there will be time allotted to discuss and evaluate the project process, track progress, discuss resource allocation, new team members. The project manager is responsible for the invite and agenda as well as managing the meetings. Agenda and discussions/decisions on these meeting will be documented and put up in the internal project web.

B.5.3.2 Project Review Workshops ("learning loops")

Every six months, as preparation for the Management Board meetings and project reviews from the EU project office, the project management team invites the team to an evaluation workshop, lasting for a day, in which product as well as process is being reviewed. Risk assessment is also an important part of this workshop. This meeting could result in proposed changes that will then be reported to the Management Board for decision. The project manager is responsible for the invite and agenda as well as managing the meetings. Agenda and discussions/decisions on these meeting will be documented and put up in the internal project web.

B.5.3.3 "Sprint" Meetings are the Key to PyPy's Technical Development

Key to PyPy's technical development and research are so called "Sprints". These publically announced one-week meetings serve as an intense working forum to rapidly discuss and implement key PyPy ideas with agile methodologies and take place on a six week cycle. The goals for each "Sprint" will be decided by the development team in cooperation with the project management team. The project manager is responsible for the handling of logistics before, during and after the sprints (invite, location, preparation etc). Agenda and discussions/decisions on these sprints will be documented and put up in the internal project web.

During the "Sprints", developers usually pair up and write unit-tests to test the to-be-implemented features before actually adding them. The unit-test-first approach helps to understand the planned feature. Additionally, the discussion in a pair makes sure that obviously wrong paths of development are avoided. If something seems too hard to test or to pin down explicitly this is taken as an indication of an underlying design problem.

Note that more traditional approaches usually follow a model where developers work alone and only meet to talk. Instead with sprint-driven development talking and actually implementing resulting ideas ensures a more focused approach with fast feedback cycles. While the free software community is successful especially because of it's open communication model sprints are an accelerator to this process. While coding in pairs developers educate each other which leads to a

Acronym: PyPy

broader common understanding of the project. During a sprint multiple pairs want to work in parallel which adds pressure on design decisions so that independent development of components of the system is possible. Thus sprints not only deepen the communication and understanding among researchers and developers but they imply a working process which enhances the software design in multiple ways.

The sprints are also the forum in which interested developers can participate in the development process. This is a fast and effective way of introducing new project members into the team, its methodologies, climate and tasks.

With a very-high-level-language like Python rapid development in coding-sprints becomes especially effective. A VHLL-language generally allows to focus on ideas rather than on low-level language details. In combination with pervasive test-driven development this eases high-quality rapid evolution towards the intended goals. Obviously, the PyPy developers are very experienced with Python and bigger applications in general. Thus the full potential of agile methodologies is unveiled during PyPy sprints. In less than five weeks worth of development (during four sprints) the group produced a working prototype which is a big success not only in the eyes of its developers.

B.5.3.4 Technical Decisions

Major design or technical decisions are usually reached through consensus during the sprints. If a conflict cannot be resolved there then the technical board gets the final say. However, it is expected that design and implementation choices will usually be determined by consensual agreement or by informal votes on the development mailing list. This is common practice within the Python and many others free software communities. Also, the PyPy developers and researchers will construct few if any formal hierarchies between them. Constantly working together with agile methodologies and the visibility of each individual contribution help enforce high-quality program code and good design decisions.

B.5.4 Quality Control of Technical Development

The PyPy project will ensure quality by a variety of means. On the grand scale, the involvement of excellent researchers ensures that the general direction takes care of latest insights in language research. Moreover, we will publish our research results on conferences and to scientific and free software communities. This forms the basis to maintain a high-quality general technical direction.

The developers deploy agile methodologies like unit-test-driven development and pair-programming. By the end of the project we expect to have produced more than 3000 unit-tests testing every aspect of the runtime system. The presence of such tests also allows to rapidly change parts of the implementation without fear of breaking functionality elsewhere. We also plan to release our runtime system often and thus gather additional feedback from early adopters, developers and researchers.

To support the open development we base all of our documents, source code and website information on a version control system. In combination with a notification on all changes this ensures that all interested parties can review and react to developments.

The PyPy developers have produced a working prototype within four one-week sprints and a little development in between. The code and design quality of the project is already widely accepted. There are now 400 unit-tests. As a consequence, Guido van Rossum, the inventor and maintainer of

Acronym: PyPy

today's Python, listed is as the number one project he would like to succeed. He previously attended one of our sprints and got deeply involved with our architecture and source code which he immediately found intuitive to work with. Thus we believe that our choices for technical quality management are fit to meet highest standards.

B.5.4.1 Additional Quality Procedures

The project manager will circulate a draft Quality Management plan for the project prior to first Project Meeting and then present it for approval at the first Meeting. It should complement the prescribed quality approach with respect to the following aspects:

- Document procedures, standards and control
- Issue control for documents
- Reporting procedures, frequency and format
- Communication procedures
- Corrective actions
- Exception control
- Conflict resolution
- Meeting draft agenda
- Format of meeting minutes
- Tracking system for actions
- Risk assessment
- Evaluation routines
- Specific responsibilities within the project

B.5.5 Communication and Reporting

The project process will be reported as follows:

- Monthly written status reports to the Management Board/Technical Board by the project management team. These reports will be posted on the internal project web for the entire team to access.
- Project review report to the EU project office. These reports are the result of the project review workshops (every 6th month) and are produced by the project management team. These reports will be posted on the internal project web for the entire team to access.
- Project evaluation report. At the end of the project, an evaluation report will be produced in which both product, process and deliverables will be evaluated. This report will be presented to stakeholders (consortium companies and partners) and the EU project office.

Transparency concerning information and decision-making is vital for the PyPy project:

- all discussions, protocols, reports, reviews and product/project information will be accessible to all people participating in the project
- information of the kind mentioned above will not be altered to suit different target groups in the project - this minimizes the risk for information to be filtered

The technical development of PyPy is driven by open continuous discussion. Many of the involved decisions are made and verified during one-week working meetings, so called "sprints". Members from the larger Python software community are publicly invited and have the chance to interact and work with the PyPy developers or become one themselves. Mailing lists, chat-sessions, Wikis and

Acronym: PyPy

notification of program changes provide a constant flow of information between PyPy project members and the wider community. Additionally, groups of developers can start interactive "screen" sessions which allows sharing their workspace and implement and communicate efficiently. Therefore conflicts out of missing or conflicting information or due to misunderstandings will be minimized.

Each sprint meeting is planned for by all developers. The sprint goals are usually agreed upon before the meeting starts. This is also important to allow new developers or contributors to join specific efforts. Sprint results are subsequently published to email and web-channels to gather feedback and educate others about changes.

We will present multiple reports and scientific papers on major conferences such as EuroPython (Python's European community conference), FOSDEM (Free and Open Source Developer European Meeting), OSCON (Open Source Convention), PyCon (Python developer conference) and to domain specific audiences such as embedded device developers. In a later phase of the project the PEP (Python Enhancement Proposals) procedures may be implemented. This is the standard procedure for applying changes to the C-implementation of Python as of today. It forces an author to clearly state the benefits of the proposed Enhancement and provides an rationale. However, such a formal method will only be required when the project reaches the point where users begin to rely on aspects of our implementation.

B.5.6 Management of Knowledge and Intellectual Property

Every contributor is fully responsible for not introducing program code which might infringe third party copyright or patents for that matter. Every contributor agrees to license his contributions under a MIT-style license approved by the Open Source Initiative and the Free Software Foundation. See section 3.

The public availability of PyPy's source code at all times on the basis on such an open and commercially exploitable license stipulates exchange of ideas, contribution to the project and reusability all parts of PyPy from the start.

In return, this provides the developers with fast feedback and improvements with respect to their current developments. At the heart of a free software community lies open communication, the free flow of information and organizing shared interests. The PyPy project is already fully involved and based on these principles. We also believe that for a language runtime system like PyPy a free license is of vital importance to reach wide deployment and recognition. Such a license is also a necessity to allow PyPy to become a reference implementation of the Python language specification.

B.5.7 Consortium Agreement

In the case that the proposal is accepted by the EU, the coordinator and the project management team will draw up a consortium agreement that formalises the license policy outlined above. As all technical deliverables will be fully public, for general modification and reuse, all partners, and indeed the rest of the world, are free to exploit the results as they choose.

B.6 Detailed Implementation Plan

B.6.1 Workplan Introduction

The PyPy project can be divided into three phases:

- Phase 1: Building a Novel Language Research Tool. The core functionality of PyPy itself must first be developed, using our novel and flexible approach of Object Spaces.
- Phase 2: High Performance. This code base can be used as a research/integration platform of choice, targeting especially performance.
- Phase 3: Validation & Flexibility. Specific applications can be implemented and disseminated.

Phase 1 is a prerequisite for Phases 2 and 3; Phases 2 and 3 are reasonably independent from each other.

Beyond phase-specific tasks, several project-long infrastructure tasks are of paramount importance. In particular, coordination is assured by the project coordinator in workpackage [WP01](#), with the help of the management and technical boards, as described in section B5. This workpackage involves collecting and monitoring monthly status reports, reporting to the EU, organising sprints, and maintaining an internal web site in collaboration with the maintenance workpackage.

B.6.2 Research and Technological Aspects and Options

B.6.2.1 Phase 1

The first phase will provide a novel architecture for research of a VHLL language. It is to be compatible with the language specification of Python and consists of the following major parts:

- A *bytecode compiler*, which inputs Python source code and outputs an internal intermediate representation, the *bytecode*.
- A *bytecode interpreter*, which interprets bytecodes and manages the supporting internal structures (frames, exception tracebacks...). It considers objects as black boxes and delegates all individual operations on them to a library of built-in types, the *Object Space*.
- An *Object Space*, which captures the semantics of the various types and objects of the language.

This subdivision is common among interpreter implementations, although we place special emphasis on the library of built-in types and its separation from the bytecode interpreter. The implementation will closely follow the current reference C implementation (CPython). It is thus expected to be relatively straightforward, varying from CPython only in a few design decisions (mainly with respect to the Object Space separation of concerns), and changing strictly nothing in the Python language itself.

This task is done in workpackage [WP04](#). After the initial development, [WP04](#) will switch its focus to reimplementing the many extension modules written in C that are either standard or widely-used in CPython. (The Python Standard Library is already mostly written in Python, so that the problem only concerns C-coded extension modules.) For each extension module, two strategies can be

Acronym: PyPy

followed: either the module is rewritten as a (regular) Python module, or it is written as an extension module of the PyPy interpreter (i.e. in Python too, but at the level of the interpreter.)

The result of [WP04](#) up to this point will still only be able to run on top of another Python implementation, such as CPython. Despite this limitation, it will still present some advantages over the existing CPython implementation, in terms of education (being a much more compact, modular and readable piece of code than CPython) and in terms of flexibility (as a basis into which experimenters can plug in alternate Object Spaces, alternate interpreters, or alternate compilers) -- more about this below.

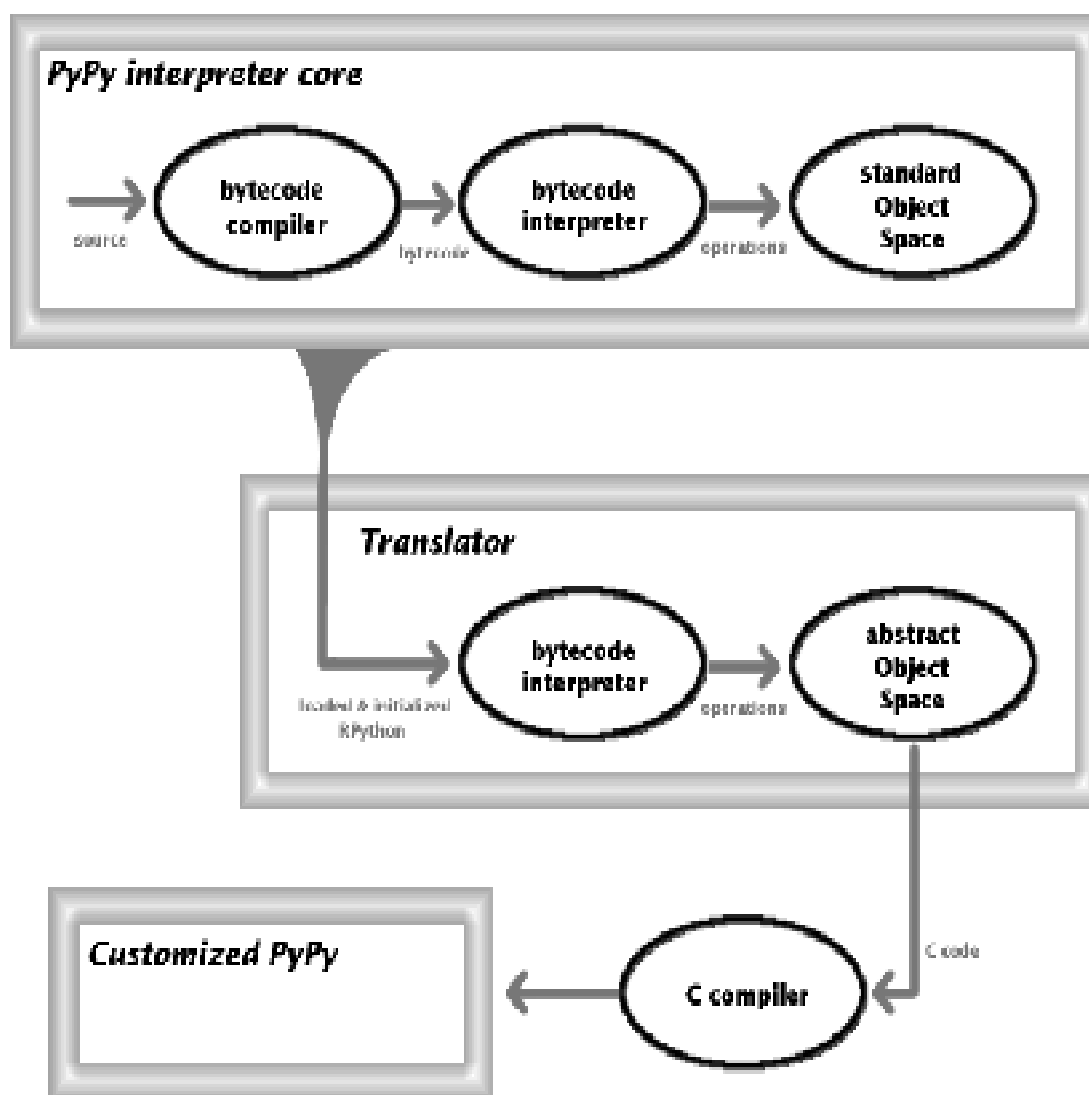
At this point, to make PyPy stand-alone (and running at a reasonable speed), we must ensure that key areas of the source have been restricted to be written in a subset of the Python language, a sublanguage (RPython) in which staticness restrictions are enforced. This sublanguage is suitable for analysis and translation into a lower-level language. The sublanguage's precise definition is a trade-off between the amount of dynamic power desired to write PyPy and the amount of effort we put in the translation tools.

Note that translation is not a one-shot process; the only source code for PyPy will be in Python or RPython, and translation can be repeated freely as part of a compilation process.

We are giving translation an innovative emphasis (and thus a whole workpackage, [WP05](#).) in the project. It is not only an RPython-to-C translator; it is a source-to-source transformer, and as such an essential part of our flexibility goals. Numerous aspects that used to be design decisions influencing the whole source code of the current CPython have by now become merely customizable behaviour of the translator. Indeed, instead of hard-coding such design decisions, we will keep the PyPy source as simple as possible, and "plug-in" the required knowledge into the translator. For example, the high-level source need not be concerned about memory management issues (garbage collection, reference counting...); this aspect can be "weaved" into the low-level code by the translator. This point is essential for the separation of concerns. Weaving aspects will be done mainly as intermediate stages, for example as transformations on the intermediate representation used by the translator, allowing control over orthogonality issues between aspects. This whole approach has deep advantages over the classic monolithic one, ranging from education (the main source base is not encumbered by details) to raw performance (choice of appropriate low-level models based on real-world context-dependent measures and comparisons). Also note this architecture's extreme adaptability: instead of generating C code, it is straightforward to target other runtime environments like Java or .NET. By contrast, today's costs of maintaining several evolving implementations (CPython, Jython for Java...) are very high.

The translation process itself requires some kind of analysis of the RPython code. Among the various ways to perform this analysis we will most probably choose the one based on *abstract interpretation*, as opposed to source-level or bytecode-level analysis:

Acronym: PyPy



The basic idea is to write an alternative "abstract" Object Space which, instead of actually performing any operation between objects, records these operations and traces the control flow. The "abstract" Object Space will be plugged into the existing bytecode interpreter; these two components, together, will then function as an abstract (or symbolic) interpreter in the usual sense of the word. The net result is that we can actually analyse RPython source code without writing any code specific to the language, given that we already have a bytecode interpreter which is flexible enough to accommodate a non-standard Object Space. In other words, the combination of PyPy and an "abstract" Object Space performs as the front-end of the translator, and can be used to translate (for example) the regular PyPy interpreter and its standard Object Space. Note the two different roles played by the bytecode interpreter in the diagram above.

Another important advantage of this approach is that, instead of operating on static source code, it works on the result of loading and initializing the code into the existing CPython interpreter. (Python, unlike more static languages, allows arbitrary computations to be performed while loading modules, e.g. initializing caches or selecting components according to external parameters.) We are thus not restricted to RPython at initialization time, which is important in order to achieve the configurability goals.

B.6.2.2 Phase 2

The completion of the first translated stand-alone PyPy interpreter is where the project could potentially branch into a large number of directions. Numerous exciting applications can be foreseen; we will examine some of them in more details in Phase 3.

Phase 2 is concerned about research, and integration of research, based on the extreme flexibility afforded by the PyPy platform.

B.6.2.2.1 Performance

Part of Phase 2 focuses essentially on performance issues, which are important in helping to establish the real-world success of a language implementation, and may open the language to a wide range of applications for which it was previously thought to be unsuitable.

The flexibility in PyPy allows a number of design decisions to be easily reconsidered; better yet, it allows different design decisions to coexist. Indeed, most "hard" issues in interpreters have no obvious best solution; they all depend in complicated ways on the specific details of the runtime environment and on the particular application considered and its possibly evolving context. PyPy will provide a good platform to experiment with, and compare empirically, several different possible approaches for many of these issues. For example:

- The language's core object types can have several implementations with different trade-offs. To experiment with this, we will write a collection of alternatives in the "standard" Object Space implementation and heuristics to select between them. This kind of research effort is common, but PyPy can provide a good platform for real-world comparisons, and to help isolate which particular choices have which effects in an otherwise unchanged environment. Such data is notoriously hard to obtain in monolithic interpreters. This is the focus of [WP06](#).
- Similarly, as described above, pervasive design decisions can be experimented with by tailoring the translator. This is the focus of [WP07](#).

We will in particular investigate in detail two specific ways to customize the translator:

- Generating Continuation Passing Style (CPS) low-level code. This makes the advanced notion of continuation available for the programmer; but -- most importantly in our case -- it allows the development, with the help of an appropriate runtime system, to support massive parallelism. Indeed, in almost any OS, native threads are not appropriate for massive usage. Applications (e.g. web servers handling thousands of connections) have to somehow emulate parallelism explicitly. Soft-threads are an ideal target for language integration. This work (also part of [WP07](#)) consists of exploiting this idea, which has been first tried for Python in the Stackless project.
- Generating a JIT compiler. Existing work in the Psyco project has shown that it would actually be possible to generate the most part of a JIT compiler instead of having to write it from scratch. The basic idea is again abstract interpretation: instead of actually performing any operation between two objects, we *generate* machine code that can perform the required operation. Again, no change to the bytecode interpreter is needed; all we need is to turn individual operation orders into processor instructions, together with a supporting runtime systems. This is defined by [WP08](#).

In dynamic languages, the truth behind JIT compiling is a bit more involved than the above paragraph suggests. All the "standard" operations in Python, including intuitively simple ones, are in

Acronym: PyPy

fact relatively complex because they depend heavily on the runtime type of the objects involved. This complex code is already written in detail in the "standard" Object Space. Thus the JIT compiler will work by abstract interpretation of RPython code, i.e. abstract interpretation of the interpreter itself (as opposed to user application code). This is similar to the ideas behind the translator, which operates on the RPython source (i.e. the bytecode interpreter and the standard Object Space). We plan to write the dynamic part of the JIT as a plug-in to the translator: instead of generating C code that is the direct translation of PyPy, we will generate C code (with the translator) that itself generates machine code (by directly emitting it as bytes into memory). This extra indirection has large benefits: the operations the JIT need to be manually taught about for the generation of machine code are only the ones allowed in RPython. In other words, we only need to design a JIT supporting the low-level operations of RPython; still, the translated piece of C code that we obtain is a JIT-enabled version of PyPy that supports the whole of the Python language.

B.6.2.2.2 Other Research Aspects

The other part of Phase 2 focuses on non-performance-oriented research-level aspects. These are the extensions enabled by the flexible modularization (bytecode compiler, bytecode interpreter, Object Spaces, and translator) enabled in Phase 1. For example, it would be possible to write an interpreter for a completely different language using the same framework, leveraging the translator and thus obtaining an efficient JIT-enabled implementation with little effort.

This is actually strongly considered as an implementation strategy for the Prolog language in a current EU-funded project:

We could replace with a reasonable amount of effort the core Python interpreter in PyPy with an experimental Prolog interpreter. This would both prove the versatility of the PyPy platform and give performance results possibly far beyond the state of the art. This is a reasonable assumption because the long history of Prolog is quite focused on static (compile-time) optimisations.

--Armin Rigo, for ASAP, Southampton team (Advanced Specialization and Analysis for Pervasive Computing), EU IST FET Programme Project Number IST-2001-38059.

In the context of the PyPy project, we will not replace the interpreter altogether, but experiment with extensions:

- based on feedback from the Python community, we will experiment with the proposed language extensions that are deemed worthy by the CPython developers. Numerous PEPs (Python Extension Proposals), for example, need an experimental implementation for testing before they can be accepted or rejected for integration into CPython; not only is PyPy a good platform to test them on, but also, playing this sandbox role will ensure PyPy remains synchronised with the development of CPython. (The role of [WP03](#) is to keep track of the ongoing development of the Python language and its CPython implementation; as a part of this task, we will also look for existing solutions to automate the "passive" part of this effort at least partially.)
- [WP09](#) involves the research and development of techniques inspired from logic programming. An inference engine already exists in the python-logic libraries. Successful integration of these concepts, far from regular imperative programming, would provide a validation of the level of flexibility we achieved in Phase 1. Similarly, [WP10](#) will experiment with OO concepts not originally present in Python, like aspect-oriented programming and design-by-contract.

B.6.2.3 Phase 3

The third phase of the project is to implement selected key applications of the flexibility we have enabled so far.

Let us stress again that Phase 3 is not meant to run only after Phase 2, but partly in parallel with it. The core flexibility on top of which we will be building is provided by the PyPy interpreter as described in Phase 1.

The applications we have selected can be categorized as follows:

- Language-level object models;
- Language-level extensions;
- Interpreter adaptations.

Phase 3 is also when we expect third parties to build on top of our platform.

B.6.2.3.1 Language-Level Object Models

This is the topic of [WP12](#). We will integrate with the language itself three middleware features: security, transparent distribution, and persistence.

Security is an important and difficult aspect that requires knowledge and support at a wide number of levels to be effective. Programming languages can help contribute to security at their level. Different object security models are possible:

- The language can be artificially restricted, with dangerous operations taken out and thus impossible to achieve by interpreted programs. In effect, this amounts to building a carefully stripped-down interpreter.
- Access to some objects can be denied to only some (untrusted) parts of the interpreted program; or access can go through proxies performing some kind of access checks. This is a more flexible solution, but it is more difficult to implement efficiently. Explicit checks for each access consume time. In PyPy this could be implemented as an Object Space checking and delegating operations to another Object Space.

For comparison, CPython used to implement security-by-absence-of-reference: untrusted parts of a program could supposedly not obtain a reference to a dangerous object. This model, called "RExec", has been recognized as fragile and hard to maintain. It has been removed from CPython.

In PyPy, one or both of the alternatives previously described can be practically implemented.

Transparent distribution of objects over a network is another middleware feature that would benefit from being implemented at the language level. This is a vast subject which has been studied extensively. There are several implementations already available in Python, with varying degrees of transparency. None however can be fully transparent by definition of the language itself, which allows introspection -- a program using introspection features could thus defeat the delicate mechanisms introduced by a network distribution library. Moreover, such libraries typically impose additional constraints, e.g. are only able to move objects over a network if they are of a class inheriting from a particular class.

We will study which of these implementations is best suited for being moved into the interpreter itself. These include transparent interfaces to remote objects (like CORBA, Java RMI...), transparent distribution of objects, and transparent distribution of processes themselves. For example, a foreseen solution for the CORBA model would be to design a Proxy Object Space that

Acronym: PyPy

delegates operations to a remote CORBA object server over the network, thus hiding the complexities of the underlying protocols. We will also study approaches enabling transparent distributed execution, i.e. in which the Object Space may decide to move, rather objects or operations, a whole algorithm's bytecode. Such totally-transparent distributed execution would support applications such as fault tolerance (e.g. perform a computation on two machines and compare the results).

Persistence is related to distribution; it is essentially a way to move objects transparently between the persistent storage and the main memory. This subject has been studied extensively, and there are several implementations already available.

We are considering at this point ZODB, the Zope Database engine, and PyPerSyst, which are quite complete. Their respective developers have already expressed high interest in PyPy. Indeed, they would benefit from better language integration; they currently put non-natural constraints on the programmer in terms of what he is allowed to do for persistence to actually work transparently. Typically, it is not possible for a library to detect changes such as adding elements to lists, so that list objects cannot be automatically marked as "dirty" (as needed to ensure they will transparently be saved when necessary). This requires language support, and more specifically an extension or proxy Object Space.

B.6.2.3.2 Language-Level Extensions

[WP09](#) and [WP10](#) are language-level extensions in the sense that they introduce new features usually present as libraries directly into the interpreter itself.

A major goal of the PyPy project is to enable people to carry out more of their programming tasks in Python without having to resort to other programming languages. We will therefore exploit the architecture and the optimisations of the PyPy implementation and the constructs for constraints and search to enhance the object model, thus offering in Python what can currently only be found in some other object-oriented languages. In particular:

[WP10](#) will offer the ability to compose program functionality with aspects and to design object-oriented programs through contract. For example, today to add aspects like logging or design-by-contract to functions you have to add a wrapper or another layer on top of existing definitions, a process which is clumsy and fragile and thus impractical. Instead it would be preferable to be able to hook into the internals of an interpreter and extend its behaviour, e.g. the method invocation mechanisms. Specifically, [WP10](#) will experiment with having first-class hooks for checking arguments and return values, or supporting efficiently "around/after and before" behaviours like the similar concepts in the Common Lisp Object System.

[WP09](#) will explore the integration of logic programming techniques. For example, today to use a constraint solver like the one from the python-logic libraries, one has to resort to an unfamiliar custom syntax. These libraries would be much more useful if they were better integrated with the language. [WP09](#) will specifically experiment with rewriting parts of the python-logic libraries in RPython for integration and performance, and enhance the syntax to support them, with the goal of supporting Semantic Web and knowledge representation applications.

B.6.2.3.3 Interpreter Adaptations

The PyPy interpreter as developed in Phases 1 and 2 will be particularly suitable to be adapted to extremely diverse runtime environments, from embedded devices to number-crunching machines.

Acronym: PyPy

In [WP11](#) we propose to study to specific case of embedded devices, which are often limited in processor speed and memory size. This either limits the power of software that is implemented for these platforms, or enforces use of low-level approaches like C/C++ or Java. PyPy is especially suited to support such platforms, since it can produce effective and compact code, while retaining the abstraction and ease-of-use of a Very-High-Level Language. Based on the specific needs of these devices, we will experiment with memory- or battery-efficient implementations of all the customizable aspects described in Phase 2. Additionally, results from [WP13](#) and feedback from one or more selected industrial partners will be used to verify the applicability of Python's flexible architecture for embedded devices in particular.

B.6.2.3.4 Integration and Configuration

It is expected that most of the applications described above will still be relatively experimental when the project ends. It is also expected that the extreme flexibility will lead to the deployment of a potentially large number of different configurations of PyPy. Thus our essential goal, beyond validating our techniques and showing what can be done and how it benefits from integration with the language, is to make this knowledge available and easy-to-use for contributors and third-parties.

To fulfill the technical aspect of this goal, we will not only release a range of different versions of PyPy with different trade-offs and including more or less of the features developed in Phase 3, but we will also build a complete toolchain to allow programmers to choose from the available features, various implementations, and runtime restrictions and build a custom PyPy version. This is the objective of [WP13](#).

B.6.2.4 Documentation and Dissemination

The documentation aspect of the dissemination goal is handled by one of the few workpackages that run for the whole duration of the project.

Each sprint, as well as the regular progress of non-sprint development, will produce technical novelties, some of which may afford immediate use and adoption of the novel technologies being developed. We expect all developers and participants to openly report to and discuss with their appropriate communities. Reports and summaries will be posted to the relevant mailing lists as well as archived on both the PyPy and the Python Business Forum website for universal availability. [WP14](#) is to support this process at all levels.

Moreover, [WP14](#) will produce a joint report about agile methodologies employed by the project. In particular, any "lessons learned" about Sprint Driven development will be analysed in cooperation with all participants, the technical and management board.

Python community members will be encouraged to keep current and get involved with the project, while community involvement and feedback on technical developments will affect design and implementation decisions. Feedback and suggestions will be gathered on the website and mailing lists and used by the appropriate project areas to further enhance the project's development efforts, in true Open Source spirit.

In addition to supporting the communication process, [WP14](#) will on occasion present longer, detailed reports to the Commission, and to scientific committees advising the Commission on technical issues. Technical papers and talks will be submitted to scientific conferences which deal with the subject matters covered by the project. When the advancement of the project warrants it, [WP14](#) will also publish "popularization" articles and tutorial materials to help other practitioners of

Acronym: PyPy

software development to make practical use of the project's results. Diagrams and schematics will be provided to illustrate fundamental concepts, as appropriate to the audience and the subject matter.

To ensure stakeholder participation and feedback that goes beyond the python community, [WP14](#) also consists of arranging workshops (6 in total) to disseminate the ongoing process- and development progress. The majority of the workshops are going to target important stakeholder groups and will primarily focus on disseminating PyPy results from an implementation perspective, customized for the target groups different needs.

B.6.2.5 Maintenance

PyPy's own development needs an infrastructure that must continuously be kept up-to-date and further developed. The role of [WP02](#) is decisive in choosing and adopting modern development environments. The main source repository is placed under control of Subversion, a Concurrent Versioning System. This workpackage also includes maintaining and possibly redesigning frameworks for unit-testing -- unit-tests are essential in the development process in sprints. The website makes heavy use of collaborative tools like Wiki pages, a content management system, and issue trackers.

Other tasks include:

- Subversion maintenance (e.g. notification hooks and pre/post-commit restrictions);
- providing access over http/https and ssh server;
- building extensions for automatic document extraction;
- maintenance or setup of mailing lists;
- implementing a search facility over all content;
- maintaining or extending the issue tracker;
- maintaining and enhancing the website infrastructure;
- performing backups for all relevant information/code;
- setting up a mirror repository which is kept up-to-date and which can be used read-only in case of failure of the main repository;
- taking care of regular repository backups, designing a strict backup policy suitable for the project, and adhering to it.

B.6.3 Risks in the Project and Steps to Minimise Them

The success of the whole project depends on the success of Phase 1. When the corresponding milestone is reached -- a complete Python interpreter in Python and a translator for it -- the project will branch. Phase 2 and Phase 3 workpackages are not tightly interconnected. So we will examine project risks with our main focus on Phase 1.

B.6.3.1 Phase 1

The initial design decisions have already been discussed during the young history of the project and its first four sprints, and a basic working prototype has already been written, as a proof of concept. We thus take for granted that the risks associated with the following decisions are almost non-existent at this point:

- writing a Python interpreter in Python is not only possible -- it is markedly faster and easier to do than in a low level language like C.
- following some of the basic design decisions of CPython (the internals of whom most of us are intimately familiar with) leads to fast development and good results.
- pluggable Object Spaces do work. Several proof-of-concept Object Spaces have been successfully tried.
- Object Spaces are a remarkably suitable abstraction for abstract/symbolic interpretation. Control flow of simple functions has been derived using an Object Space.

Moreover, simple type analysis and generation of low-level code from a low-level representation is common in a variety of contexts, so we don't expect any particular problem from that issue either.

As a fall-back solution, we are aware of the more common way to do the translator analysis: if the abstract Object Space should fail to scale as expected, we will revert to classic bytecode-based analysis, adapting tools that are already available for this task. We will, however, pursue the abstract interpretation solution as far as possible, because it leads to more language-independence and particularly to a single place where all knowledge about the details of the actual bytecode needs to be represented.

B.6.3.2 Phase 2

Phase 2 is mainly research-oriented. It is by definition more difficult to predict the involved risks, and fall-back strategies if risks materialize, for this Phase than for the others.

However, a risk-mitigating factor is that the performance goals of Phase 2 are backed by the existing and working prototypes Stackless and Psyco, whose authors are among the PyPy members. Further, it is also important to note that the described performance goals, while very interesting and important in order to widen the application area of the language, are not essential to the other applications we have planned. Phase 1 (including translation) already produces a level of performance that can reasonably be expected to be comparable to the existing CPython interpreter.

B.6.3.3 Phase 3

Phase 3 consists of a number of independent applications. Consequently, a failure in one of them would almost certainly not influence the others. Of course, appropriate effort has nevertheless been taken to avoid failure in each of the mentioned applications: all of them are to some extent already

Acronym: PyPy

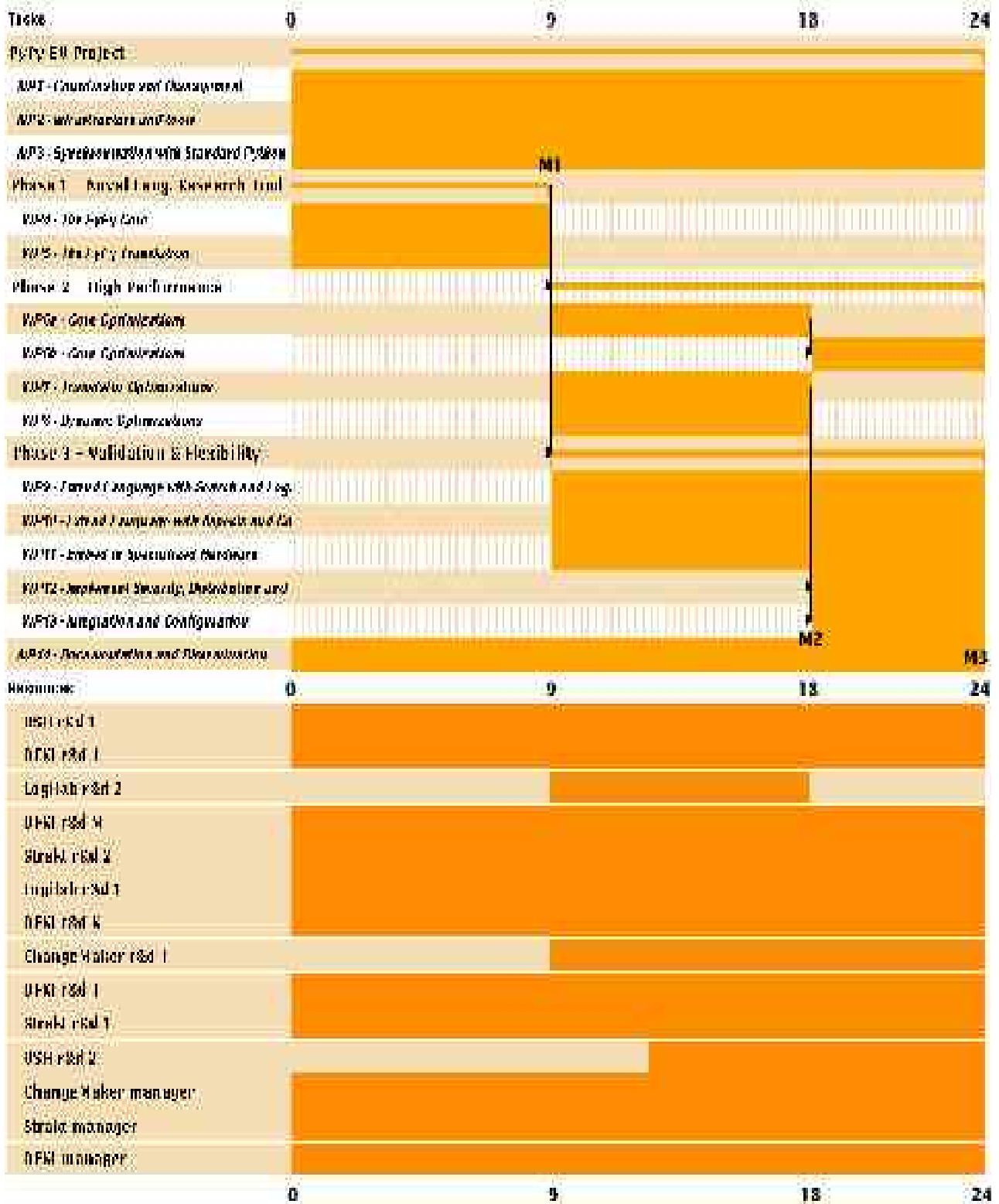
existing as middleware libraries, and people fluent in the corresponding domains generally appreciate the advantages a better language integration would bring.

The project concludes with an Integration and Configuration workpackage; should one of the previous applications fail, the final configurable tool will still be able to integrate the others. The only risk involved there would involve difficulties (technical or others) to integrate the work done by the various partners. As a fall-back solution we may have to keep some successful but hard to integrate applications out of the mainstream full-featured release or even configuration tool. It is a tendency inherent in open source projects to sometimes fork, in order to explore alternative possibilities, and later, when consensus is reached, merge again. We will try to minimize the corresponding risk (of divergence, i.e., a fork not followed by a later merge), but we also think that it is inherently very low, as long as the different applications are kept as non-overlapping as possible. This risk minimization strategy in application-structuring is already reflected in the workpackages presented for Phase 3.

Acronym: PyPy

B.6.4 Project Planning and Time Table (Gantt chart)

Acronym: PyPy



B.6.5 Workpackage List

Work-package No	Workpackage title	Lead contractor No	Person-months	Start month	End month	Deliverable No
WP01	Coordination and Management	1 (DFKI)	12	0	24	
WP02	Infrastructure and Tools	1 (DFKI)	8	0	24	
WP03	Synchronisation with Standard Python	2 (USH)	9	0	24	
WP04	The PyPy Core	3 (Strakt)	23	0	9	
WP05	The PyPy Translation	2 (USH)	29	0	9	
WP06	Core Optimisations	2 (USH)	13	9	24	
WP07	Translator Optimisations	1 (DFKI)	18	9	18	
WP08	Dynamic Optimisations	3 (Strakt)	25	9	18	
WP09	Extend Language with Search and Logic	1 (DFKI)	19	9	24	
WP10	Extend Language with Aspects and Contracts	4 (Logilab)	12	9	24	
WP11	Embed in Specialized Hardware	5 (CM)	6	9	24	
WP12	Implement Security, Distribution and Persistence	3 (Strakt)	13	18	24	
WP13	Integration and Configuration	1 (DFKI)	12	18	24	
WP14	Project Documentation and Dissemination	5 (CM)	23	0	24	

B.6.6 Deliverables List

Deliverable No	Deliverable sortformtitle	Deliverable Date	Nature	Dissemination Level
D01.1	Create QA plan for the project	1	O	PU
D01.2	Collection and monitoring of reports	24	O	PU
D01.3	Report to EU	24	O	PU
D01.4	Organise meetings	24	O	PU
D01.5	Organize sprints	24	O	PU
D01.6	Create and maintain internal web site	24	O	PU
D01.7	Newsletter for external stakeholders	24	O	PU
D02.1	Configuring, installation of all development tools	24	O	PU
D02.2	A release scheme for PyPy versions	24	O	PU
D02.3	Http-server to present runtime/introspection	24	P	PU
D02.4	Several website revisions	24	O	PU

Acronym: PyPy

Deliverable No	Deliverable sortformtitle	Deliverable Date	Nature	Dissemination Level
D02.5	Automated (unit-)testing framework	24	P	PU
D03.1	A PyPy that supports a subset of the CPython API	24	P	PU
D03.2	Report on supporting full C-extensions compatibility	24	R	PU
D03.3	A guide describing the porting of C extensions	24	R	PU
D04.1	Partial Python Implementation on top of CPython	6	P	PU
D04.2	Complete Python Implementation on top of CPython	9	P	PU
D04.3	Report about the parser and bytecode compiler	10	R	PU
D04.4	Release PyPy as a research tool	9	P	PU
D05.1	Publish on translating a very-high-level description	9	R	PU
D05.2	A compiled, self-contained version of PyPy	9	P	PU
D05.3	Publish on implementation with translation aspects	10	R	PU
D05.4	Publish on encapsulating low level language aspects	10	R	PU
D06.1	Integrate the first core optimisation results	18	P	PU
D06.2	Publication of statistics for core objects	24	R	PU
D06.3	Publication of novel heuristic techniques	24	R	PU
D07.1	Support for massive parallelism	18	P	PU
D07.2	Publish optimisation results	19	R	PU
D07.3	Report about practical usages	18	R	PU
D07.4	Report on approaches for translator aspects	19	R	PU
D08.1	A processor back-end supporting Intel(tm) i386	18	P	PU
D08.2	A processor back-end supporting PowerPC	18	P	PU
D08.3	A Just-In-Time compiler for PyPy	18	P	PU
D08.4	Publish a report about the new JIT architecture	19	R	PU
D08.5	Publish a research report comparing JIT techniques	19	R	PU
D09.1	Initial constraint satisfaction and inference engine	18	P	PU
D09.2	Improved constraint satisfaction and inference engine	24	P	PU
D09.3	Assessment of benefits	24	R	PU
D10.1	Aspect-oriented programming capabilities in PyPy	24	P	PU
D10.2	Design-by-contract capabilities	24	P	PU
D10.3	Advanced static checking capabilities	24	P	PU

Acronym: PyPy

Deliverable No	Deliverable sortformtitle	Deliverable Date	Nature	Dissemination Level
D11.1	Written report to Axis Communication	21	R	PU
D11.2	Documented feedback from Axis Communication	24	R	PU
D12.1	Prototype supporting control of individual operations	24	P	PU
D12.2	Prototype supporting transparent remote proxying	24	P	PU
D12.3	Prototype supporting orthogonal persistence	24	P	PU
D12.4	Report on novel directions in middleware features	24	R	PU
D13.1	A release with all optimisation and runtime features	24	P	PU
D13.2	A build- and configuration tool	24	P	PU
D13.3	A publication about PyPy code base and customization	24	R	PU
D14.1	Report about Milestone/Phase 1	10	R	PU
D14.2	Tutorials and a guide through the PyPy source code	15	R	PU
D14.3	Report about Milestone/Phase 2	19	R	PU
D14.4	Report about Milestone/Phase 3	25	R	PU
D14.5	Documentation of the development process	24	R	PU
D14.6	Organize four domain specific Workshops	24	O	PU
D14.7	Arrange two nondomain specific workshops	24	O	PU
D14.8	Participation of PyPy members in conferences	24	O	PU

B.6.7 Workpackage description

B.6.7.1 Coordination and Management

Workpackage number	WP01	Start date or starting event:	0
Participant id	CM	DFKI	STRAKT
Person-months per participant:	4	4	4

Objectives

Coordination and Management of the PyPy Project.

Description of work

- collection and monitoring of monthly status reports,
- reporting to the EU project office
- organising project meetings and sprints
- maintaining contents and structure of the project web site. The website will make heavy use of collaborative tools like Wiki pages, a content management system, and issue trackers.

Deliverables

- D01.1 Create QA plan for the project
- D01.2 Collection and monitoring of reports (monthly, sprints, review workshop, evaluation)
- D01.3 Report to EU
- D01.4 Organise project meetings
- D01.5 Organize sprints
- D01.6 Create and maintain internal web site
- D01.7 Create and maintain newsletter covering ongoing project process for external stakeholders

Milestones and Expected Result

A succesful and measurable delivery of every workpackage within the project scope.

B.6.7.2 Infrastructure and Tools

Workpackage number	WP02	Start date or starting event:	0
Participant id	DFKI		
Person-months per participant:	8		

Objectives

Supporting the PyPy project by producing and enhancing the tools for version control of source code, communication, dissemination and distribution of information. Collaborate with community to implement a powerful development environment for the various needs of the PyPy STREP.

Description of work

Support the development process with reusing and furthering existing or developing new debugging opensource tools.

- maintain the Subversion repository including notification hooks and pre/post-commit restrictions.
- provide access over http/https and ssh to the repository.
- setup support for releasing different versions of PyPy
- build extensions for automatic document extraction
- maintain/setup mailing lists
- implement a search facility over all content
- maintain/extend issuetracker
- maintain and enhance website infrastructure
- perform backups for all relevant information and sourcecode
- setup a mirror repository which is kept up-to-date and which can be used readonly in case of failure of the main repository.
- Take care of regular repository backups. Design a strict backup policy suitable for the project, and adhere to it.
- setup and maintain mailing lists for development, consortium and community communication.
- help with automated testing

Deliverables

- D02.1 configuring, installation of all development tools (subversion, apache2, mailman, backup, roundup, ...)
- D02.2 introduce and document a release scheme for PyPy versions
- D02.3 http-server to present runtime/introspection information aiding debugging and understanding of PyPy internals
- D02.4 several website revisions
- D02.5 release automated (unit-)testing framework with html/pdf reports

Milestones and Expected Result

Provide advanced infrastructure to support goals and objectives of the project.

B.6.7.3 Synchronisation with Standard Python

Workpackage number	WP03	Start date or starting event: 0
Participant id	USH	DFKI
Person-months per participant:	3	6

Objectives

Keeping PyPy in sync with potential changes to Standard Python. Support a subset of the CPython API for compatibility with extension modules. Facilitate porting of extension modules to PyPy. Collaborate strongly with CPython developer communities.

Description of work

Task 1

Monitor the development of Standard Python for its ongoing changes. Check the relevance of the changes concerning the structure of PyPy. Collaborate with CPython developer communities. Embed the relevant changes with proper testing, notifying the other concurrent tasks of these changes. Investigate tools which could partially automate integration of simple changes.

Task 2

Investigate different solutions to address compatibility with existing C extensions. Collaborate with CPython developer communities. Support a subset of the CPython API. Facilitate porting of more involved extensions. Investigate adjusting existing interface generators (in particular SWIG, used e.g by wxPython, subversion bindings and other popular extensions) to target PyPy.

Deliverables

- D03.1 A version of PyPy that supports a subset of the CPython API
- D03.2 A report on the feasibility of supporting full C-extensions compatibility
- D03.3 A guide describing the porting of C extensions to PyPy

Milestones and Expected Result

PyPy is kept in sync with potential changes to Standard Python throughout the duration of the project.

B.6.7.4 The PyPy Core

Workpackage number	WP04	Start date or starting event: 0	
Participant id	Strakt	DFKI	Logilab
Person-months per participant:	6	12	5

Objectives

Building a complete Python interpreter written in Python, using a subset of Python that avoids dynamic features which would impair the objectives of [WP05](#) (RPython). Collaborate with CPython core developers.

Description of work

Task 1

Implement an interpreter that is able to accept the complete Python language specification, built according to the general modularity goals described in the body of the text. This task specifically includes research and implementation work leading to:

- the Object Space interface.
- a bytecode interpreter, accepting the standard CPython bytecode format.
- a "standard" object space implementing the core Python objects' normal semantics.

This task excludes the standard extension modules and the parser and bytecode compiler, which for the purpose of testing are borrowed from the underlying CPython interpreter on top of which our interpreter runs. Collaborate strongly with CPython core developers via the python-dev mailing list.

Task 2

Port the standard Python "builtin" library:

- types (there are some 100 of them);
- built-in functions;
- built-in modules;
- other built-in objects, e.g. exception classes.

Research and decide on a case-by-case basis how to implement each one within PyPy in a simple and efficient way. The common options are to either re-implement them inside the interpreter where appropriate, or to provide a pure Python replacement (which in some cases already exists and can be borrowed).

(contd....)

Description of work (continued)

Task 3

Complete the interpreter with a Python source parser and bytecode compiler. Collaborate with various core developers involved with new parsing and compiling techniques.

- Implement a Python parser and bytecode compiler in Python by leveraging existing research and implementation work. Reuse code and ideas from ongoing open source projects.
- Design a flexible way to insert the compiler into the code base, easing future experimentation with the syntax.
- More generally, research and provide bridges between the interpreter source and the code it interprets, allowing the interpreter to delegate some of its "core" work to regular, interpreted, pure Python code. Decide when it is appropriate to make this regular code visible to language users (e.g. as modules, modifiable at run-time).

Deliverables

- D04.1 First partial Python Implementation running on top of CPython
- D04.2 Complete Python implementation running on top of CPython
- D04.3 Report about the parser and bytecode compiler implementation
- D04.4 Release PyPy as a research tool for experimental language enhancements

Milestones and Expected Result

- A partial Python implementation that passes 90% of the official Python test suite for core language features, i.e. not depending on library-level C extension modules (the C part of the "standard library").
- Providing a substitute for the many existing C modules will be an ongoing effort necessarily involving the wider python communities. By the end of Phases 2 and 3 of the project we expect to have the most commonly used modules reimplemented in a flexible framework or - if technically not feasible - matched with equivalent functionality in PyPy.

B.6.7.5 The PyPy Translation

Workpackage number	WP05	Start date or starting event: 0	
Participant id	STRAKT	DFKI	USH
Person-months per participant:	15	6	8

Objectives

Analysis and translation of the PyPy core ([WP04](#)) into efficient low-level code (C, Pyrex, Java, others). Providing a Runtime Library for the translated versions of PyPy.

Description of work

Task 1

At the core of this task lies the research and implementation of full "abstract interpretation" by reusing the unmodified PyPy interpreter with a special object space.

- Create a code analysis tool for a subset of the Python language (RPython).
- Coordinate the definition of RPython with [WP04](#), being the implementation language for most of the core.
- Experiment with different ways of doing Abstract Interpretation. Look for an Abstract Domain suitable for type inference. Compare with published algorithms for type inference. Select and implement the most appropriate solution in our context.

Task 2

Produce a tool chain, capable of extracting the RPython bytecode from the core, translating it into low-level code (with C being the primary target) and compiling it.

- Create a build process for statically generating and running a low-level PyPy interpreter and object space. The LLVM (Low level virtual Machine) project is a candidate for collaboration in this context.
- Provide hooks into internals to alter translation aspects.
- Research/experiment with a Java backend.

Task 3

In order to give a working environment to the translated RPython program, build the low-level-specific runtime components of PyPy. For the C PyPy runtime, important parts can be directly re-used from CPython. However, certain aspects such as memory management and threading models are to be implemented in a modular way.

- Implement a minimal C-runtime for the translated PyPy, collaborate for with CPython core developers.
- Integrate Memory Management, threadings models, and other aspects in a modular way into both the C-runtime and the statically generated PyPy low level code. For memory management improve and build on top of solutions from other open source language projects.

Acronym: PyPy

Deliverables

- D05.1 Publish a report about translating a very-high-level description of a language into low level code by building on "abstract interpretation" and PyPy's separation of the interpreter and the object space in particular.
- D05.2 Release a compiled, self-contained modular static version of PyPy.
- D05.3 Publish solutions and open challenges regarding the implementation of memory management/threading models as translation aspects.
- D05.4 Publish an overview paper about the success of encapsulating low level language aspects as well defined parts of the translation phase.

Milestones and Expected Result

- M1 Complete implementation of the Python language, conforming to the language definition and passing all language compliancy* - tests of the official Python test suite.
- Research Results on encapsulating low level language aspects as parts of a translation from very-high-level into low-level code.

*Some of the the many hundred CPython language tests actually test implementation details like e.g. memory management. The exact line between a language feature and an implementation detail is at times hard to draw. In case of doubt the original CPython developers will be consulted for clarification.

B.6.7.6 Core Optimisations

Workpackage number	WP06	Start date or starting event: 9
Participant id	USH	DFKI
Person-months per participant:	12	1

Objectives

Building upon the efficiency and flexibility of the code base developed in [WP04](#), investigate and compare alternative designs and implementations.

Description of work

Task 1

Provide alternative implementations of the core objects, such as dictionaries, strings and integers, with different trade-offs (speed, size, limitations). Collect existing techniques from the literature and research new ones.

Identify other performance-critical parts of the PyPy source and provide alternative implementations (new or from the literature).

Task 2

Run performance tests on the above, comparing the different implementation decisions for a range of platforms and applications. Categorize into overall improvement, platform, and application dependency. Produce and publish reports on the results.

Task 3

Merge the results back into the optimisation effort. Where necessary, define heuristics to select implementations and to switch between them, depending on evolving runtime conditions. Collect existing heuristics from the literature and research new ones. Report on the results and submit as publications.

Deliverables

- D06.1 Integrate the first core optimisation results into the PyPy core at the end of phase 2
- D06.2 Publication and analysis of performance statistics for different implementations of core objects
- D06.3 Publication of novel heuristic techniques

Milestones and Expected Result

- Measurably better performance for non-trivial programs.

B.6.7.7 Translator Optimisations

Workpackage number	WP07	Start date or starting event: 9
Participant id	DFKI	USH
Person-months per participant:	15	3

Objectives

Identification and Implementation of Optimisations through modifications of the Translator. Enable Massive Parallelism in a Single Thread. Provide support for soft-real-time parallelism. Allow Pickling of a Running Program.

Description of work

Task 1

Implement memory-efficient massive parallelism complementing the threads based on C stack switching provided by Oses or semi-OS-independent libraries.

- Enhance the translator to support continuation passing style by integrating technology from the Stackless project.
- Implement the necessary runtime system to support massive parallelism Minimize the resource footprint of each "microthread".
- To complement explicit (application-defined) scheduling, implement a pre-emptive scheduler at the bytecode level, with priorities, distributing "microthreads" to one or a small number of OS threads.
- Implement pickling a running program or a selected microthread (i.e. serializing continuations).

Task 2

Use PyPy as a research tool for general optimisations.

- Study approaches concerning code size vs. speed trade-offs.
- Implement and compare different object layout and memory management strategies. Implement schemes of pointer tagging.
- Enhance multimethod dispatching and other places where hand-crafted optimisations can help the translator.
- Create reports and merge the results back into the optimisation effort. As of general interest, submit the reports as publication.

Deliverables

- D07.1 Release a version of PyPy that supports automatic or explicitly scheduled massive parallelism
- D07.2 Publish optimisation results
- D07.3 Report about practical usages of massive parallelism and program pickling
- D07.4 Report on approaches for memory management, object implementations and other translator aspects for high performance PyPy

Milestones and Expected Result

- M2 High performance PyPy, together with [WP08](#). Outperform the state-of-the art (Psyco, Stackless).

B.6.7.8 Dynamic Optimisations

Workpackage number	WP08	Start date or starting event: 9	
Participant id	STRAKT	DFKI	USH
Person-months per participant:	17	2	6

Objectives

Enhance PyPy to dynamically adapt to its run-time environment and to the characteristics of the running program. Dramatically increase speed by enabling Just-In-Time compilation and specialization. Address multiple processor architectures.

Description of work

Task 1

Make a Just-In-Time compiler for the PyPy framework.

- Apply and enhance techniques from the Psyco project. Combine them with the static type inference techniques implemented in [WP05](#).
- Leverage the static translator to generate instrumenting and self-specializing code for all the parts of the PyPy interpreter and standard Object Space that support possibly time-critical application code.
- Implement the run-time management environment to support the JIT compiler (in-memory machine code and supporting data structures, profiling and statistics collection, etc).

Task 2

Make the JIT compiler flexible and portable.

- Enable dynamic foreign function calls. Allow language users to use them in their applications. Collaborate strongly with core developers from the ctypes and SWIG open source projects.
- Design and implement a back-end component for dynamically emitting machine code for multiple processor architectures.

Task 3

Research optimisation heuristics for the Just-In-Time compiler.

- Research by comparing existing and novel techniques for JIT compilers heuristics, management issues, etc.
- Coordinate with [WP06](#).

Deliverables

- D08.1 A processor back-end supporting Intel(tm) i386
- D08.2 A processor back-end supporting PowerPC
- D08.3 Release A Just-In-Time compiler for PyPy
- D08.4 Publish a report about the new JIT architecture, its performance and how its techniques can be applied to languages other than Python/PyPy
- D08.5 Publish a research report comparing JIT techniques

Milestones and Expected Result

- M2 High performance PyPy, together with [WP07](#). Outperform the state-of-the art (Psyco, Stackless). Verify the expectation of reaching half the speed of C for purely algorithmic code.

B.6.7.9 Extend Language with Search and Logic

Workpackage number	WP09	Start date or starting event: 9
Participant id	DFKI	Logilab
Person-months per participant:	10	9

Objectives

Leveraging PyPy flexibility implement language-integrated constraint satisfaction algorithms and inference engine to allow logic programming for Semantic Web applications developed at Logilab and DFKI.

Description of work

Task 1

Using the flexible architecture provided by the PyPy interpreter, we will first reimplement the current python-logic libraries available from Logilab to better integrate with the language and gain important execution speed-ups.

Task 2

We will investigate alternative implementation techniques to improve the efficiency of the search component. These will build on ideas from constraint languages such as Oz/Mozart and, where appropriate, will take advantage of the new avenues for implementation opened up by the object space architecture.

Task 3

This logic programming enabled Python interpreter will then be used to support frameworks for Semantic Web applications that are on-going at Logilab and DFKI and which are related to RDF-based approaches to knowledge representation.

Deliverables

- D09.1 Initial implementation of constraint satisfaction engine and inference engine based on Logilab's library.
- D09.2 Improved implementation of constraint satisfaction engine and inference engine in PyPy. Benchmarks will be provided, to measure performance improvements.
- D09.3 Assessment of benefits obtained from using PyPy over current tools to further develop Semantic Web projects at Logilab and DFKI, accompanied by appropriate code examples.

Milestones and Expected Result

- Python interpreter exhibiting logic-programming features, such as inference and constraint satisfaction.

B.6.7.10 Extend Language with Aspects and Contracts

Workpackage number	WP10	Start date or starting event: 9
Participant id	Logilab	DFKI
Person-months per participant:	9	3

Objectives

Leveraging PyPy flexibility implement aspect-oriented programming, design-by-contract and advanced static checking capabilities.

Description of work

Task 1

Using the flexible architecture provided by the PyPy interpreter, we will first reimplement the current aspect-oriented libraries available from Logilab, to better integrate with the language and greatly simplify the design and enhance the performance.

Aspect-Oriented Programming is a technique that improves on Object-Oriented Programming by facilitating the division of concerns in the implementation of classes. Some behaviors may be made into aspects that are later weaved with the existing implementation of classes for which one wants to add the aspect's behavior. An example would be a Logger aspect weaved with a class to provide logging functionality without the class having any knowledge of logging.

Logilab already published an implementation of aspects for Python, but it is limited by the language in the sense that weaving code consists of wrapping methods instead of actually producing new code or modifying existing code. This adds overhead and other constraints that would be removed with a PyPy-based implementation. Once the equivalent of Logilab's aspect library will be implemented in PyPy, further progress will be made using articles and research tools from www.aosd.net, the aspect-oriented software development community and research portal.

(contd...)

Description of work (continued)

Task 2

Implement design-by-contract using the aspect-enabled interpreter.

Design-by-contract, as first exhibited by the Eiffel object-oriented programming language, can be implemented as a specific aspect as was done by Logilab in its aspect library. A PyPy-based implementation would allow for further experiments in terms of syntax improvements on the contract definition, but also better performance.

Task 3

Implement advanced static checking and code correctness checking capabilities, thus furthering the work done with the pychecker and pylint tools.

Static checking tools for Python already exist, but still have difficulties at the moment when it comes to type checking and inference. The implementation of static checking in PyPy will benefit from PyPy's runtime optimisation mechanism based for part on type inference, but also from articles describing the implementation of type checking and type inference for other languages and from existing implementations of pychecker and Logilab's pylint tools.

Deliverables

- D10.1 Implementation of aspect-oriented programming capabilities in PyPy
- D10.2 Implementation of design-by-contract in PyPy
- D10.3 Implementation of advanced static checking capabilities

Milestones and Expected Result

- Python interpreter exhibiting aspect-oriented programming, design-by-contract and advanced static checking capabilities.

B.6.7.11 Embed in Specialized Hardware

Workpackage number	WP11	Start date or starting event: 9
Participant id	Logilab	CM
Person-months per participant:	5	1

Objectives

Axis Communications (<http://www.axis.com/>) of Lund, Sweden have shown an interest in how PyPy could be implemented on their hardware and have specifically asked for information regarding this part of the PyPy project (WP13). Axis is a world leading company in printer servers and network video products. Change Maker, who have an ongoing relation with the company will be an interface between Axis.

The objective is to write a specific report targeting embedded software companies with the results from [WP13](#) (usage of PyPy custom version and the build- and configuration tool). This report will showcase usage with Axis hardware as an example.

Description of work

Task 1

Write report to Axis showcasing usage of PyPy custom version. Report and analyse feedback from Axis regarding content of report. Communicate to Axis all reports and build tools developed in [WP13](#).

Deliverables

- D11.1 Written report to Axis Communication
- D11.2 Documented feedback from Axis Communication

Milestones and Expected Result

- Dissemination and feedback on the PyPy custom version and the build- and configuration tool from Axis Communication, a target stakeholder within the embedded sector.

B.6.7.12 Implement Security, Distribution and Persistence

Workpackage number	WP12	Start date or starting event: 18
Participant id	STRAKT	USH
Person-months per participant:	10	3

Objectives

Research and validate the flexibility and accessibility of PyPy by building experimental prototypes of key middleware features into the language itself, namely, security, remote execution, and orthogonal persistence. Publish reports outlining the novel future directions opened up by the PyPy architecture, building on the experimental support provided.

Description of work

Task 1

Build an experimental prototype on security by controlling the individual operations performed by a non-trusted program. Coordinate with [WP14](#) and consult with researchers from the IBM Zurich Research Lab in particular as to which level of control is needed. Report on the novel future directions.

Task 2

Build support for transparent remote proxying at the language level. Look up existing libraries. Typical libraries like the Zope Enterprise Objects (ZEO) or CORBA/Java RMI models require programs to be aware of the remote execution model; by contrast, experimentally build the support for an equivalent functionality directly into the language. Collaborate with existing open source projects. Report on the novel future directions.

Task 3

Build persistence at the language level by implementing an orthogonally persistent object space for Python programs. Look up existing libraries. Persistence is never fully orthogonal without advanced language support, as witnessed by libraries like the Zope Database (ZODB); we will build an experimental object space that can provide full orthogonally. Collaborate with ZODB and PyPersyst developer communities. Report on the novel future directions.

Deliverables

- D12.1 An experimental prototype supporting security through control of individual operations at the language level.
- D12.2 An experimental prototype supporting transparent remote proxying at the language level.
- D12.3 An experimental prototype supporting orthogonal persistence at the language level.
- D12.4 Publish a report outlining the novel future directions opened up by the PyPy architecture in middleware features.

Milestones and Expected Result

- M3 Validation of the flexibility of PyPy with respect to key middleware requirements

B.6.7.13 Integration and Configuration

Workpackage number	WP13	Start date or starting event: 18
Participant id	DFKI	Logilab
Person-months per participant:	8	4

Objectives

Integrate research and the source code from [wp06](#), [wp07](#), [wp08](#), [wp09](#) and [wp10](#). Analyse remaining problems. Develop tools that will allow to chose from available features and runtime restrictions to build a custom PyPy version.

Description of work

Task 1

Analyse and integrate all results from the results of other workpackages.

- Provide a consistent and combined code base for translation aspects, the JIT-compiler and language extensions
- assert benefits and open problems regarding interaction between the different translation and language extension aspects.
- write a report about the combined PyPy code base.

Task 2

Implement user interfaces to select features and runtime restrictions.

- Provide a way to automatically build a PyPy custom version for different memory, performance and extension requirements.
- Research and select defaults for environments with large and with small amounts of system resources.

Deliverables

- D13.1 A release of PyPy with all available optimisation and runtime features
- D13.2 A build- and configuration tool that allows to build custom PyPy versions suitable for specialized environments like small or very large computing devices
- D13.3 A publication about the combined PyPy code base and its customization possibilities.

Milestones and Expected Result

- A stable release of PyPy enabling a wide range of language users to experiment and develop using it as a flexible and highly-optimizable Python implementation.

B.6.7.14 Documentation and Dissemination

Workpackage number	WP14	Start date or starting event: 0
Participant id	CM	DFKI
Person-months per participant:	11	12

Objectives

Providing ongoing documentation throughout the whole project. Supporting the dissemination process at all levels. Analysis of "Sprint driven development" and the agile development methods of the project. Report to the Commission in collaboration with the PyPy consortium.

Description of work

Task 1

Support the dissemination process at all levels.

- Write, disseminate and archive reports about each development sprint and produce newsletter
- Gathering feedback from participants, present a detailed joint report about agile methodologies employed by the project.
- Ensure that all reports and publications that are being produced as part of workpackages 1-14 are effectively disseminated to python and non-python communities.

Task 2

Provide longer, detailed reports to the commission and scientific committees advising the Commission on technical and agile development issues.

Task 3

When the advancement of the project warrants it, publish "popularization" articles and tutorial materials to help other practitioners of software development to make practical use of the project's results.

- Diagrams and schematics will be provided to illustrate fundamental concepts, as appropriate to the audience and the subject matter.
- Tutorials will be published on the web site and disseminated to various communities
- Utilizing tools from [WP02](#) a guide through the source code of PyPy will be provided.

(contd...)

Description of work (continued)

Task 4

Prepare and organize workshops with special interested groups at all levels (the PyPy interpreter, the build toolchain to create custom interpreters, third-party language extensions or alternate languages, and theoretical aspects). Take part in conferences. Interact with various python and non-python communities.

- For example, Reseachers at the IBM Zurich Resarch lab have shown interest in our work and volunteered to discuss and peer-review our published results. We plan to present our security findings during a joint workshop and solicit non-confidential input from the research group on Identity Management and Privacy (Dr. Matthias Schunter).
- Generally reach out to companies and research groups inside and outside the python community by organizing workshops to disseminate project results and gather feedback with respect to our novel language architecture. 2 specific workshops will be arranged in this manner (during phase 1 and at the end of phase 3).
- organize Workshops during the course of the PyPy STREP to allow for adaptation of detail planning on specific feedback issues from specific target groups (industrial companies, game developing companies and SME:s).

Deliverables

- D14.1 Report about Milestone/Phase 1
- D14.2 Tutorials and a guide through the PyPy source code
- D14.3 Report about Milestone/Phase 2
- D14.4 Report about Milestone/Phase 3
- D14.5 Documentation of the development process and "Sprint driven development" in particular
- D14.6 Organize four domain specific Workshops during phase 3 with various business and research communities. Reports about these workshops.
- D14.7 Arrange two nondomain specific workshops disseminating development process and results (during phase 1 and phase 3)
- D14.8 Participation of PyPy researchers and developers in conferences throughout the project process.

Milestones and Expected Result

Good documentation and dissemination to multiple business and research communities through promotion, workshops and conference. Useful feedback on the practical flexibility of our language research efforts.

A better understanding how research and development can be accelerated through agile methodologies and "Sprint Driven Development" and how the PyPy project achieved its goals.

B.7 Other Issues

B.7.1 Ethical Considerations

1. Proposers are requested to fill in the following table

Does your proposed research raise sensitive ethical questions related to:

Human beings	No.
Human biological samples	No.
Personal data (whether identified by name or not)	No.
Genetic information	No.
Animals	No.

2. Proposers are requested to confirm that the proposed research does not involve:

Research activity aimed at human cloning for reproductive purposes	No.
Research activity intended to modify the genetic heritage of human beings which could make such changes heritable.	No.
Research activity intended to create human embryos solely for the purpose of research or for the purpose of stem cell procurement including by means of somatic cell nuclear transfer;	No.
Research involving the use of human embryos or embryonic stem cells with the exception of banked or isolated human embryonic stem cells in culture	No.
Confirmation : the proposed research involves none of the issues listed in section B	YES.

There are however, other ethical issues which reflect the philosophy of the Open Source Community. We believe that it is unethical to withhold the source of fundamental tools upon which other people rely. When a tool that is critical to your business or your daily life fails, it becomes vital for you to get it fixed. Either you can fix it yourself, or you can pay somebody else to fix it for you. If that vital tool is proprietary, then you are immediately stuck in a Market situation. If the only people who can fix the tool work for high wages in a country with a high standard of living, then all their dependent customers who live in poorer conditions are at a disadvantage, often an unsurmountable one. For lack of the financial resources to pay for needed fixes, small businesses fail or produce software that performs poorly in the marketplace, riddled with bugs that cannot be addressed. This, in turn, prevents those poorer companies from attaining success. They are marginalised, all unintentionally, by the normal workings of the Market. They become a second-tier 'user' of the product, dependent on their software provider to 'do the right thing', even though 'the right thing' may vary enormously from region to region.

Thus, when you deny people the option of being able to 'do it yourself' you contribute to the forces in society which produce wealth-based social differentiation and segregation. 'The rich get richer and the poor get poorer'. A compiler is an extremely fundamental tool which one uses to create any

Acronym: PyPy

sort of software which one can think of. This is precisely the sort of tool that must be freely available to the rich as well as the poor, in all its entirety. The poorer nations will then be in a non-dependent situation. As they use their imaginations and creativity and labour to increase their own personal standard of living and that of the communities where they live, they will have one fewer thing to worry about -- what will I do if my compiler has a bug?

B.7.2 Gender Issues

Women are chronically (and increasingly) underrepresented in the computer field; particularly among designers of software and products.

Studies^{1 2} have shown several factors involved in this gender gap, among them:

- Lack of experience with computers leading to avoidance of CS as a field of study or employment.
- Interest in "computing for a purpose" rather than "programming for programming sake"
- Focus on usability and usefulness

PyPy addresses these concerns in the following ways:

Allowing the creation of mobile devices which are easily programmed will increase women's successful experiences with computers, thus increasing their comfort level and likelihood of pursuing education and careers in computers. The ease of creating programs due to PyPy's simple but powerful language will allow women to incorporate programming in pursuit of their interests. The user-friendly, interactive interfaces available will attract more women into programming and designing software.

As more women bring their viewpoints to the design-process, better products in terms of meeting the needs of a broader number of people will be created and increase the appeal of the products among women, which in turn will encourage women to use and familiarize themselves with the new technologies.

Furthermore, the increase in economic and educational opportunities afforded in the CS fields must not be ignored. Studies have shown that, while women who do enter CS enter with less experience, they demonstrate no less ability than their male counterparts. By breaking down the barriers to entry noted above, PyPy will help lessen the gender gap, technologically, socially and economically.

As an example of the beneficial effects that Python already has on introducing women to the field of programming, the Georgia Institute of Technology teaches a course for non-Computer Science

¹Becoming a computer scientist: a report by the ACM committee on the status of women in computing science. Authors: Pearl, Amy; Pollack, Martha E.; Riskin, Eve; Thomas, Becky; Wolf, Elizabeth; Wu, Alice. Journal: Communications of the ACM, Nov. 1990, v33, n11, p47-58.

²Unlocking the Clubhouse: Women in Computing Allan Fisher and Jane Margolis, PIs School of Computer Science Carnegie Mellon University MIT Press, 2002, ISBN 0262133989.

Acronym: PyPy

majors which focuses on media, rather than the traditional math-oriented model. This is the Abstract of the paper³ describing the course:

Computing may well become considered an essential part of a liberal education, but introductory programming courses will not look like the way that they do today. Current CS1 course are failing dramatically. We are developing a new course, to be taught starting in Spring 2003, which uses computation for communication as a guiding principle. Students learn to program by writing Python programs for manipulating sound, images, and movies. This paper describes the course development and the tools developed for the course.

While only 3 out of 14 expected participants in the project are female, this merely reflects the poor state of gender equality in the field of programming. Indeed, the ratio in this project is actually more than twice of what is normal.

B.7.3 Safety Issues

Nobody who uses a computer can avoid noticing how viruses, unsolicited email and computer attacks have made life difficult for computer users. While PyPy is unable to solve such problems, it may mitigate them. There are 3 factors that affect this:

1. The ease of writing Python programs give the developers more time to consider security implications.
2. Python as a language has built in checks against buffer overflows and other classic problems that make traditional languages like C and C++ prone to attacks. By improving Python performance, a larger set of programs will be immune to such exploits.
3. PyPy provides the tools for adding specialised security features to the language, in the form of new Object Spaces. This allows applications where security is paramount to add checks and partitioning that will significantly strengthen the defenses. As far as we know, no other language has this feature.

B.7.4 Other Policy Related Issues

As we mentioned before, one of the greatest threats to European competitiveness is its dependence upon proprietary closed source software, mostly made in the United States. The short-term threat of dependence upon your supplier was already discussed in section 3.

There is another threat, which is more insidious, and more long term. **A good workman knows his tools**. This is much more than the theoretical knowledge of how his tools ought to work, according to principles learned in school. The way car mechanics know how cars work is distinctly different from what you would know if you had attended classes on 'the principles of the internal combustion engine'.

Right now, in Europe, we don't have enough of the software equivalents of car-mechanics. And most of them live in academia, where they know the intimate details of languages that never get used in industrial applications. They are the Formula-One race car mechanics of the software world.

³ <http://coweb.cc.gatech.edu/mediaComp-plan/uploads/37/ITICSE-mediacomp2.pdf>

Acronym: PyPy

While race car mechanics do serve a useful purpose, we have a much greater need for people who know how to repair the family car.

It is not as if there is a shortage of people who would be interested in learning such things, if the source were made available. Many people have taken this step by learning how CPython does its stuff. But still there is a barrier. If you want to know how CPython does things, you need to learn C. C is a notoriously difficult language to learn.

But let us quote an article posted to the Python-in-Education mailing list in its entirety:

Date: Sun, 14 Sep 2003 11:52:05 -0400
From: Arthur <ajsiegel@optonline.net>
To: edu-sig@python.org
Subject: [Edu-sig] re : If the PyPy Project ...

List-Id: Python in education <edu-sig.python.org>

Terry -

>Since I presume the goal of PyPy is to implement *Python* in Python,
>wouldn't the implementation language be rather insignificant to an
>end-user such as an educator? Why would it be "better" than CPython?

For whatever reason, the complex built_in and the cmath module, implemented in Python, are part of the early PyPy codebase. As I had been spending some time in the complex realm with PyGeo - a simple version of the complex realm, as these things go - Laura's post gave me the impetus to try to plugin the PyPy implementations.

Only got stuck on the typing issue. My code tests for instance(object,complex). The PyPy complexobject, unadorned, is a class - and fails the test. But that leads me into a deeper look at some of the PyPy codebase, trying to understand a little bit of how this kind of issue are to be dealt with. Not that I got there, yet - but I did seem to have an avenue to explore I would not have with CPython - as someone who doesn't C, and has no intention of trying, seriously, to do so.

As someone living within the limits of having Python as my only real language, I think that PyPy should open things up for me considerably. It will make Python, I believe, a more attractive educational language, because it will make someone with a strong foundation in Python - as the language of choice - a more self-sufficient programmer.

Presumably - the point is - there will be less cases where the right approach would be an extension module in C or C++, and a sense of fundamental compromise should one not be equipped to go there. Many thousands of folks - using VB and the like - already do involved, highly performing real world applications and make nice livings doing so, without being equipped to do C. I am thinking that PyPy would put Python more squarely in that "space".

Is any of this so, or just hope?

Art

Here is somebody who is hoping we can give him a language he can understand and use without learning C. He is the author of PyGeo, a dynamic geometry laboratory and toolkit, commonly used by elementary and high school teachers. This is where the future lies. Python is already an excellent teaching language. PyPy will be a better one.

Acronym: PyPy

Thus PyPy comes full circle, and back to Python's original heritage, as a language that was designed to be 'easy to use' for non-programmers. In the appendix, a proposal *Computer Programming For Everybody*, written in 1999 by Guido van Rossum, Python's creator, is as relevant now as when it was written. To summarise:

Python's predecessor, ABC, was designed at CWI in the early eighties as a teaching language. Its motto was "stamp out Basic" -- the main competition in languages for non-experts at the time. ABC's designers had a lot of experience teaching traditional programming languages like ALGOL to novices. They discovered that students were often so overwhelmed by the incidental details of using a computer language they never managed to focus on good program and algorithm design.

ABC's designers therefore set out to design a language that would take care of all the incidentals, leaving the student more time to learn about programming, and not the chore of 'how to use a computer'. They proposed both a new language design and new terminology that deviated radically from what was (and still is) current among computer scientists and programmers. This proved to be a barrier towards widespread adoption. ABC was too different. People actually wanted a language which was more like other languages, not one that simply didn't relate to what programmers would be doing the rest of their lives. Thus ABC remained a teaching language, and nothing more.

About a decade later, Python grew out of this frustration. It shares ABC's focus on elegance of expression, fundamentals of programming, and taking away incidentals, but adds the modern tools that programmers have come to expect, such as object-orientation, and a large and powerful standard library.

The PyPy project, being a focused STREP, does not intend to address the problem of 'Computer Literacy' and 'Computer Programming For Everybody'. But we believe that these problems would make an excellent candidate for a Python-related Specific Supporting Action, and we would make every effort to aid and assist such an action should the Commission call for it.